KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science

TERMINATION ANALYSIS OF CONSTRAINT HANDLING RULES: THEORY AND PRACTICE

Paolo PILOZZI

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

November 2011

TERMINATION ANALYSIS OF CONSTRAINT HANDLING RULES: THEORY AND PRACTICE

Paolo PILOZZI

Jury:
Prof. Dr. ir. H. Van Brussel, chair
Prof. Dr. D. De Schreye, promotor
Prof. Dr. B. Demoen
Prof. Dr. ir. F. Piessens
Prof. Dr. ir. T. Schrijvers
(U.Gent)
Dr. A. Serebrenik
(T.U.Eindhoven)

Dissertation presented in
partial fulfillment of the
requirements for the degree
of Doctor in Engineering

November 2011

D/2011/7515/137
ISBN 978-94-6018-436-9

Dankwoord

Dit proefschrift, *Termination analysis of Constraint Handling Rules: Theory and practice*, bevat de belangrijkste resultaten van mijn onderzoek naar de analyse van eindigheid in de programmeertaal Constraint Handling Rules. Dit werk werd uitgevoerd binnen de groep Declaratieve Talen en Artificiële Intelligentie van het Departement Computer Wetenschappen aan de Katholieke Universiteit te Leuven. Zonder hulp had dit werk niet tot stand kunnen komen. Het is dan ook vanzelfsprekend om de eerste pagina's van dit manuscript te wijden aan hen die ik er dankbaar voor ben.

Eerst en vooral zou ik de promotor van mijn doctoraat, Prof. Dr. Danny De Schreye, willen bedanken. Vooreerst om het gestelde vertrouwen, zodat ik het doctoraat heb mogen aanvangen. Belangrijker nog, om zijn begeleiding tijdens mijn doctoraat, op vlak van communicatie en onderzoeksmethodologie, wat van mij een betere onderzoeker heeft gemaakt.

Ook zou ik de leden van mijn jury willen danken voor het lezen en beoordelen van dit proefschrift. Ik zou graag Prof. Dr. ir. H. Van Brussel willen danken voor het willen voorzitten van mijn jury, Prof. Dr. D. De Schreye voor het willen promoten van dit werk, de assessoren, Prof. Dr. B. Demoen en Prof. Dr. ir. F. Piessens, voor het willen opvolgen van dit werk en de externe juryleden, Prof. Dr. ir. T. Schrijvers (U.Gent) en Dr. A. Serebrenik (T.U.Eindhoven), voor het willen zetelen in mijn jury. Hun opmerkingen waren essentieel om tot dit werk, in zijn uiteindelijke vorm, te komen.

Meer specifiek zou ik Bart Demoen —een man die zelfs met vogeltjes kan praten— willen bedanken. Eerst en vooral voor de CHR-lunchen die mij toelieten om recente ontwikkelingen te bespreken en die mij inzage gaven in de problematieken van anderen binnen de onderzoeksgroep. Dit was een leerrijke ervaring, dat nuttig is gebleken in de ontwikkeling van mijn onderzoek. Verder wil ik hem danken voor zijn intiatief en inzet betreffende de Prolog Programming Contest, een fakkel die hij recentelijk doorgaf aan Tom Schrijvers.

Dit brengt mij bij Tom Schrijvers. Het was hij die mij als kind in aanraking heeft gebracht met de eerste programmeertaal, GWBasic, die ik heb leren kennen. Het ontdekken van een programmeertaal heeft in grote mate mijn interesse naar computer architecturen bepaald en heeft uiteindelijk geleid tot het aanvangen van een doctoraat waarin ik het gedrag van declaratieve programmeertalen zou bestuderen. Het was dan ook een leuke verrassing om hem opnieuw tegen te komen als toenmalig onderzoeker binnen dezelfde groep. Opnieuw bracht Tom mij in contact met een programmeertaal, ditmaal Constraint Handling Rules. Deze ontdekking was bepalend voor de keuze van mijn onderzoeksonderwerp en heeft uiteindelijk geleid tot dit schrijven. Verder, ben ik hem ontzettend dankbaar voor zijn hulp in de beginfase van mijn onderzoek, wat resulteerde in mijn eerste publicatie.

Graag wil ik nog een aantal mensen binnen de onderzoeksgroep danken. Vooreerst wil ik Maurice Bruynooghe danken voor zijn inzet in de organisatie van de groep Declaratieve Talen en Artificiële Intelligentie. Het was een voorrecht met hem een paper te mogen schrijven. Ook wil ik graag Gerda Janssens danken voor haar uitleg over haar werk in programma-analyse, maar ook voor de vele leuke babbeltjes die mijn dag goedmaakten. Verder wil ik mijn directe collega's: Leslie De Koninck, Stef De Poorter, Jon Sneyers, Peter Van Weert, Dean Voets en Pieter Wuille; danken voor de vele discussies, al dan niet werk-gerelateerd, en voor de leuke tijden op conferentie.

Finally, I would like to thank some people outside the department that I had the opportunity of meeting during my research. First and foremost, I would like to thank Ole Torp Lassen. As an everlasting optimist, it has been a pleasure spending time with him. Also, I would like to thank Wim Vanhoof and his students for the enjoyable times we had together. Furthermore, I would like to thank John P. Gallagher, just for being a wonderful person.

Vervolgens wil ik nog het Fonds voor Wetenschappelijk Onderzoek en het Instituut voor Wetenschap en Technologie danken voor de financiële ondersteuning en het in mij gestelde vertrouwen dat daarmee gepaard gaat.

Dit brengt mij bij die mensen, in mijn persoonlijke levenssfeer, zonder wie dit werk zeker ook niet mogelijk had geweest.

In eerste instantie wil ik graag mijn ouders, Antonio Pilozzi en Agnes Vandessel, bedanken. Het zijn zij die mij de voedingsbodem gaven om uit te groeien tot de persoon die ik vandaag ben. Er zijn geen woorden om te omschrijven hoe dankbaar ik jullie hiervoor ben. Ook wil ik mijn doop-peter -en meter, Mimo Pilozzi en Franca Donadeo, en mijn vormsel-peter -en meter, Stefaan Terryn en Bernadette Henry, bedanken voor de ondersteuning die ze mijn ouders hebben geboden in het creëren van die voedingsbodem.

Ik wil ook graag mijn broer, Michael Pilozzi, en tevens beste vriend, bedanken. Eerst en vooral om zich oprecht te interesseren in het werk van zijn broer, hoe abstract het ook moge lijken. Maar ook om het te willen begrijpen, om het zo vervolgens te kunnen ridiculiseren. En, misschien wel het allerbelangrijkste, om mij te doen grijnzen terwijl ik dit schrijf.

Ik wil ook graag Thierry Dupont, Mattias Hutsebaut, Kristof Martens, Bruno Tacquenier, Yannick Thiry, Joris Vander Cammen, Els Vanhoutte en Dean Voets danken voor hun steun in moeilijke tijden, voor de geladen avonden op café, voor de vele feestjes, voor de muziek, voor de kalme momenten in gezelschap, voor de hevige discussies en voor nog zoveel meer.

Als laatste wil ik mijn verloofde, Barbara De Bolle (a.k.a. the Lady of the Spoon), bedanken. Het staat als een paal boven water, dat had ik haar niet tegengekomen, dit werk niet gerealiseerd had geweest. Dank u om er elke dag voor mij te zijn. Ik hou van u.

Paolo Pilozzi, November 2011.

Abstract

Constraint Handling Rules (CHR), closely related to Logic Programming (LP), is a declarative programming language. Over the years, the language proved successful for implementing many kinds of problems efficiently. Mainly this, but also its simple syntax and semantics, accounts for its success and impact on the research community.

To further encourage the use of CHR, we need to further improve its efficiency of execution. To this end, termination analysis of CHR can be seen as one of the main sources of input. Furthermore, in the context of program debugging, termination analysis of CHR is an important asset. Due to the many language specifics, it is often hard for programmers to point out unwanted loops in their CHR programs. It is therefore essential to have a good understanding of the termination problem in CHR.

Until recently, however, there was only an informal discussion on termination of the subset of CHR that only considers simplification rules. The contributions of this thesis are therefore twofold. First, we provide for a theoretical framework for termination analysis of the full CHR language. Secondly, based on this theoretical framework, we derive an approach for automated termination analysis of CHR. This approach extends the approaches in LP to integer polynomial interpretations and can be modularised. Furthermore, the approach is practical, as we demonstrate with T*CoP, a Termination analyser for CHR on top of Prolog.

Beknopte samenvatting

Constraint Handling Rules (CHR) is een declaratieve programmeertaal, sterk gerelateerd aan Logisch Programmeren (LP). Doorheen de jaren, bewees de taal haar nut voor het efficiënt implementeren van vele soorten problemen. Voornamelijk dit, maar ook de eenvoudige syntax en semantiek van de taal, is de reden voor haar succes en impact op de onderzoeksgemeenschap.

Om het gebruik van CHR verder te bevorderen, moet de uitvoeringsefficientie van de taal verder verbeterd worden. Daartoe is eindigheidsanalyse van CHR programma's één van de belangrijkste bronnen van informatie. Maar ook in de context van programma-debugging is eindigheidsanalyse voor CHR een belangrijke aanwinst. Door de vele taal-specifieke eigenschappen is het voor programmeurs vaak moeilijk om de oneindige lussen in hun CHR programma's te duiden. Het is daarom essentieel het eindigheidsprobleem in CHR goed te begrijpen.

Tot voor kort was er echter enkel een informele discussie rond het eindigheidsgedrag van CHR programma's waarin er enkel simplificatie mag optreden. De bijdragen van deze thesis zijn daarom tweeledig. Eerst voorzien we de volledige CHR taal van een theoretisch kader voor de analyse van eindigheid. Dan, hierop gebaseerd, voorzien we de taal van een automatiseerbare aanpak. Deze aanpak breidt de technieken in LP uit naar integer polynomische interpretaties en kan gemodulariseerd worden. Bovendien is de aanpak praktisch, zoals we aantonen met T*CoP, een werktuig voor de automatische eindigheidsanalyse van CHR programma's.

Contents

List of Figures

List of Tables

Abbreviations

CHR	Constraint Handling Rules
CHR(Prolog)	CHR on top of Prolog
CT	Constraint Theory
LP	Logic Programming
non-selfs	non-self-sustainable
RC	Ranking Condition
SCC	Strongly Connected Component
selfs	self-sustainable
SLD	Selective Linear Definite (clause)
T*CoP	Termination analyser for CHR on top of Prolog

List of Symbols

$(\geq, >)$	standard reduction pair on $\mathbb{N}$
$(\succeq, \succ)$	reduction pair (on $Call(P, I)$)
$(\succeq_P, \succ_P)$	reduction pair on $Atom_P$
$(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$	reduction pair on $Atom_P^{\mathbb{N}}$
$(\succeq_{\mathbb{P}}, \succ_{\mathbb{P}})$	reduction pair on $Call(P, I)$ w.r.t. $\mathbb{P}$
$(\succeq_\mu, \succ_\mu)$	multiset extension of a reduction pair
$(\succeq_\omega, \succ_\omega)$	reduction pair on the CHR state representation
$(\succeq_\pi, \succ_\pi)$	reduction pair on the CHR state representation for propagation
$(\succeq_\tau, \succ_\tau)$	reduction pair on tokens
$(R, \overline{c})$	token with labeled constraints $\overline{c} = c_1 \sharp i_1, \ldots, c_n \sharp i_n$
$=$	equality
$[\overline{e}]$	ordered list of elements $\overline{e} = e_1, \ldots, e_n$
$[]$	empty list
$[X \mid Y]$	to represent $[X] \bowtie Y$
$\approx$	equivalence relation of $\succeq$
χ	substitutions related to RC
$\cup$	set union
$\emptyset$	empty (multi)set
$\mathbb{N}$	set of positive integer numbers

$\mathbb{P}$	polynomial interpretation
$\mathbb{P}^s$	symbolic polynomial interpretation
$\mathbb{Z}$	set of integer numbers
$\bowtie$	list concatenation
$\langle S, m_S \rangle$	multiset, with underlying set S and multiplicity function m_S
$\langle S, T \rangle_\nu$	theoretical CHR state
$\langle U, \mu_S \rangle$	multiset, with universe U and multiset indicator function m_S
$\|c\|_{\mathbb{P}}$	level mapping of a constraint c w.r.t. $\mathbb{P}$
$\|c\|^s_{\mathbb{P}}$	symbolic level mapping of a constraint c w.r.t. $\mathbb{P}^s$
$\|Q\|^q$	abstract CHR state representation of Q
$\|Q\|^q_\omega$	CHR state representation of Q
$\|Q\|^q_\pi$	CHR state representation for propagation of Q
$\|t\|_{ll}$	list-length of a term t
$\|t\|_{ts}$	term-size of a term t
$[\![\overline{e}]\!]$	multiset of elements $\overline{e} = e_1, \ldots, e_n$
$[\![]\!]$	empty multiset
$\mathcal{C}$	abstract CHR constraint
$\mathcal{D}_P$	the CHR dependency graph of P
$\mathcal{I}n_P$	the set of abstract input constraints of P
$\mathcal{I}n_R$	the set of abstract input constraints of R
$\mathcal{L}$	first order language underlying P
$\mathcal{M}_P$	match transition relation of P
$\mathcal{N}_P$	the CHR net of P
$\mathcal{O}ut_P$	the set of abstract output constraints of P
$\mathcal{O}ut_R$	the set of abstract output constraints of R
$\mathcal{R}$	decreasing ranks set of a strict multiset decrease

$\mathcal{T}_P$	rule transition relation of P
$\mathfrak{R}_{c/n}$	polynomial interargument relation for c/n
$\mathfrak{R}^s_{c/n}$	symbolic polynomial interargument relation for c/n
$\mathfrak{R}^s_{c/n}(\bar{t})$	symbolic interargument relation for c/n w.r.t. $\bar{t} = t_1, \ldots, t_n$
$\mathfrak{R}^{ASs}_{c/n}$	symbolic added set relation for c/n
$\mathfrak{R}^{ASs}_{c/n}(\bar{t})$	symbolic added set relation for c/n w.r.t. $\bar{t} = t_1, \ldots, t_n$
$\mathfrak{R}^{AS}_{c/n}$	added set relation for c/n
$\mathfrak{R}^{CSs}_{c/n}$	symbolic call set relation for c/n
$\mathfrak{R}^{CSs}_{c/n}(\bar{t})$	symbolic call set relation for c/n w.r.t. $\bar{t} = t_1, \ldots, t_n$
$\mathfrak{R}^{CS}_{c/n}$	call set relation for c/n
$\mathfrak{R}^{QSs}_{c/n}$	symbolic query set relation for c/n
$\mathfrak{R}^{QSs}_{c/n}(\bar{t})$	symbolic query set relation for c/n w.r.t. $\bar{t} = t_1, \ldots, t_n$
$\mathfrak{R}^{QS}_{c/n}$	query set relation for c/n
$\mathfrak{R}^{SSs}_{c/n}$	symbolic success set relation for c/n
$\mathfrak{R}^{SSs}_{c/n}(\bar{t})$	symbolic success set relation for c/n w.r.t. $\bar{t} = t_1, \ldots, t_n$
$\mathfrak{R}^{SS}_{c/n}$	success set relation for c/n
$\|t\|_{\mathbb{P}}$	norm of a term t w.r.t. $\mathbb{P}$
$\|t\|^s_{\mathbb{P}}$	symbolic norm of a term t w.r.t. $\mathbb{P}^s$
ν	label assigned to constraint added next
$\bar{t}$	abbreviates a sequence of terms $t_1, \ldots, t_n$
$\overline{X}$	abbreviates a sequence of variables $X_1, \ldots, X_n$
ϕ	substitutions in computations
ϕ^q_i	accumulated substitutions in computations
$\Pi_{\mathbb{N}}$	the set of polynomials with positive integer coefficients

$\Pi_{\mathbb{Z}}$	the set of polynomials with integer coefficients
ρ	decreasing rank of a strict multiset decrease
$\rightarrow_P$	transition relation for P
$\setminus$	operator for set difference (or to separate kept and removed head)
$\sharp S$	cardinality of a (multi)set S
σ	match substitution
$\sqsubseteq, \sqsubset$	multisubset, strict multisubset relation
$\subseteq, \subset$	subset, strict subset relation
$\succ$	strict partial order relation of $\succeq$
$\succ_{\mathbb{P}}$	strict partial order on $Atom_P \cup Term_P$ induced by $\mathbb{P}$
$\succ_{\mathbb{Z}}$	strict partial order on polynomials of $\Pi_{\mathbb{Z}}$
$\succeq$	pre-order relation
$\succeq_{\mathbb{P}}$	pre-order on $Atom_P \cup Term_P$ induced by $\mathbb{P}$
$\succeq_{\mathbb{Z}}$	partial order on polynomials of $\Pi_{\mathbb{Z}}$
θ	answer substitution
$\uplus$	multiset join
$\vee$	disjunction
$\wedge$	conjunction
$\wp(\mathcal{C})$	to obtain the denotation of $\mathcal{C}$
$\xrightarrow{\square}$	to denote in computations that an answer state has been reached
$\xrightarrow{CT}$	to denote transitions in computations due to the CT
$\xrightarrow{P,\phi}$	to denote transitions in computations of P with substitutions ϕ
$\xrightarrow{P}$	to denote transitions in computations of P
$\xrightarrow{R}$	to denote transitions in computations due to a rule R
$\{\overline{e}\}$	set of elements $\overline{e} = e_1, \ldots, e_n$

$\{\}$	empty set
$Add(P, I)$	added set of P for I
$Atom_P$	constraints of P
$Atom_P^{\mathbb{N}}$	labelled constraints of P
B_j	j-th body built-in constraint of a rule R
B_j^i	j-th body built-in constraint of the rule R_i
B_P	Herbrand base of P
$c\sharp i$	constraint c labelled with i
$c(\overline{t})$	constraint $c(\overline{t})$ with terms $\overline{t} = t_1, \ldots, t_n$
C_j	j-th body CHR constraint of a rule R
C_j^i	j-th body CHR constraint of the rule R_i
$Call(P, I)$	call set of P for I
$con(c\sharp i)$	to obtain the constraint c
$Const_P$	constants of P
CT	constraint theory defined in the host language
D_j	j-th body constraint of a rule R
Fun_P	functor symbols of P
G^i	to represent the conjunction $G_1^i \wedge \cdots \wedge G_{k_i}^i$ of guards of R_i
G_j	j-th guard constraint of a rule R
G_j^i	j-th guard constraint of the rule R_i
H_j	j-th head constraint of a rule R
H_j^i	j-th head constraint of the rule R_i
I	query set
$id(c\sharp i)$	to obtain the label i
in_j^i	j-th abstract input constraint of the i-th rule
$M_{(i,j,k,l)}$	match transition relating out_j^i and in_l^k

out_j^i	j-th abstract output constraint of the i-th rule
P	CHR or CHR(Prolog) program
p/n	predicate p/n with predicate symbol p and arity n
$Pred_P$	predicate symbols of P
Q	CHR state
R	name of a rule that is used as a reference to the rule
R_P^{SSE}	extended success set of P
R_P^{SS}	success set of P
$R_{p/n}^{SSE}$	extended success set of a predicate p/n
$R_{p/n}^{SS}$	success set of a predicate p/n
$rel(c)$	to obtain the predicate of a constraint c
S	constraint store (or used as arbitrary set or multiset)
T	propagation forecast (token store)
T_i	rule transition of R_i
$T_{(D,S)}^A$	added tokens when adding constraints D to S
$T_{(D,S)}^E$	eliminated tokens when removing constraints D from S
$Term_P$	terms of P
$Token_P$	tokens of P
U	propagation store (or used as arbitrary universe)
U^1	the first-layer propagation of U
U^A	constraints added to a propagation store
U^R	constraints removed from a propagation store
$U^{1'}$	everything in U except for U^1
U^{A_1}	the first-layer propagation of U^A
U_P	Herbrand universe of P
Var_P	variables of P

Chapter 1

Introduction

In this chapter, we give an extensive introduction into the several aspects of *termination analysis of Constraint Handling Rules* (CHR). First, we position and motivate our research in the field of computer science. Then, we outline the termination problem in CHR, focussing on the particularities of the language and the similarities with other programming languages. Finally, in Section 1.4, we provide an overview of the thesis.

1.1 Motivating termination analysis

Termination analysis of declarative programming languages is an actively researched topic in computer science [DM79, Der87, AB90, AP91, DSD94, AP94, LSS97, Stä98, CT99, DDSV99, Frü00, OCM00, DLSS01, GC05, BCER02, LJBA01, DSS02, SDS03, GTSKF04, SDS04, SDS05b, SDS05a, MB05, BCG+07, GSKT+07, EWZ08, NDSGSK11, SKGN10]. This is motivated by various reasons [DSD94].

Especially in early work, this research was strongly related to *control generation* and *systematic program development*. In the context of Logic Programming (LP), building on the "*Algorithm = Logic + Control*" equation [Kow79], systematic program development can be seen as a two-phased process. First, provide for a correct logic specification of the problem domain. Then, obtain a suitable control for the specification to obtain a correct and efficient program.

A minimal requirement for a suitable control is that the program should terminate for any query of interest. As a result, termination analysis has been used to provide one of the basic sources of input for control generation.

In more recent work, additional motivations have been related to *improved precision of program analysis* given the termination properties of a program. One example is *determinism (confluence) analysis* with applications in control generation as well. Another is *equivalence analysis* with applications in program transformation. Still another is the general setting of *abstract interpretation*.

Automating correctness proofs has been another important motivation for termination analysis. Proving a program's correctness involves proving that either the program always returns a correct answer in a finite number of steps, or to prove that a program will never stop running while satisfying certain properties.

Yet another motivation for termination analysis work is to get a better understanding in the *decidability of the termination (halting) problem* for subclasses of programs. Most important in this work is to identify the boundary between minimal subclasses of programs which still have the expressibility of a Turing Machine and maximal classes for which the halting problem is decidable.

A final motivation can be related to *program debugging*, where termination analysis can provide programmers with useful information concerning unwanted loops or unexpected termination of the program.

1.2 Motivating termination analysis for CHR

CHR [Frü98, SVWSDK10, Sch08] is a declarative programming language, related to LP [Llo87, Apt90], constraint logic programming [JM94, JMMS98, MS98] and concurrent committed-choice logic programming languages [SR90]. Other influences for CHR have been the chemical abstract machine [BCLM88], Term Rewrite Systems (TRS) [BN98] and production rule systems [SS96].

CHR is a (concurrent) committed-choice logic programming language, intended for implementing custom constraint solvers [Frü98]. Over the years, the language proved also successful for implementing many other kinds of problems efficiently [SVWSDK10]. Mainly this, but also CHR's simple syntax and semantics, accounts for its success and impact on the research community.

To further encourage the use of the programming language CHR, we need to further improve the efficiency with which CHR programs are executed.

Therefore, we need more accurate program analyses [Sch05, SVWSDK10]. To this end, CHR termination analysis can be seen as one of the main sources of input. That is, the termination properties of a CHR program can be used to improve the precision of other analyses that are useful to obtain more efficient CHR programs. Examples of this are abstract interpretation [SSD05], confluence analysis [AFM96, Abd97] and equivalence analysis [AF99].

A second motivation is to get a better understanding of the termination problem in CHR. As we will explain in the next sections, there are some interesting features of CHR —unique to the language— that make termination analysis for CHR hard. For similar reasons, it can often be difficult for programmers to point out unwanted behaviour of their programs. Therefore, a third motivation can be related to program debugging.

1.3 Defining the termination problem for CHR

Termination analysis, as a field of study, is mainly concerned with the development of systematic approaches for proving termination of programs. These approaches are preferably easy to automate. Termination analysis is however an undecidable problem for Turing complete programming languages like CHR [Sne08], and thus no single systematic approach can be developed, successful on all programs. Therefore, the development of termination proof methodologies, successful on most programs that occur in practice, is the main concern of the field.

It is not surprising that most work on termination analysis has been done for declarative programs. After all, for imperative programming languages, such as Java and C, the control is specified by the programmer, who is therefore more directly responsible for unwanted loops, and —to some extent— the efficiency of his programs. This contrasts the declarative case, where termination properties are necessary for generating efficient control and locating programming errors.

Mostly for LP [AB90, AP91, BCF91, DSD94, BCER02], TRS [DM79, Der87], and their variants (see e.g., [VDS01, MR03, PM09, Wal07, SKGS+10]), termination has been studied thoroughly. Many techniques for automating termination proofs resulted from this work [AP94, LSS97, Stä98, CT99, DDSV99, OCM00, GC05, LJBA01, SDS03, SDS04, GTSKF04, SDS05b, SDS05a, MB05, GSKT+07, BCG+07, EWZ08, NDSGSK11, SKGN10]. It is therefore interesting to investigate the applicability of these techniques to CHR.

Relating CHR-termination to LP-termination:

1. The atoms of both a CHR program and an LP program are first-order relations on terms. Thus, the notions of *reduction pairs* [NDSGSK11, SKGN10] from LP are relevant to CHR.
2. *CHR simplification* and LP resolution are similar. Thus, conditions on transition rules can be related to decreases on program states, much in the same way [DSD94, DSS02].
3. Both CHR and LP allow for non-ground queries, of which the variables can get instantiated during program execution. Thus, the notions of a *call set* and *rigidity* of the call set w.r.t. a reduction pair [DSD94, DDSV99, NDSGSK11] are relevant to CHR.
4. Both CHR and LP allow for intermediate calls in rules. Thus, the use of *interargument relations* to estimate the effect of these calls [DSD94, DDSV99, NDSGSK11] is relevant to CHR.

But, CHR has also many unique features:

1. It is usually built on top of a *host language*.
2. Its rules do not act on goals of sequentially ordered atoms, but on multisets of constraints (*constraint stores*).
3. It has *multi-headed guarded rules* that only fire on a *matching* multiset of constraints (in the store) for which the guard is satisfiable.
4. In addition to multi-headed variants of LP-clauses (*simplification rules*), it also supports *propagation rules*, which do not remove any constraints from the constraint store. These rules only add new constraints.

In the remainder of this chapter, we discuss a number of language-specific issues concerning CHR-termination, on an intuitive level. We hope to provide the reader with the main intuitions, before addressing those issues in technical detail in the next chapters. Note that in doing so, we only discuss CHR-termination in relation to LP-termination since CHR-termination is only related to a lesser extent to TRS-termination.

1.3.1 Transformational vs. direct approaches

There are two distinct approaches to handle termination analysis of a programming language. In a *transformational approach*, the program to be proven terminating is transformed to a programming language in which the termination problem is better understood. As such, existing techniques, developed for other programming languages, are made available to the programming language at hand. *Direct approaches* are concerned with techniques that are directly formulated on the program to be proven terminating. Such approaches provide a better insight in the termination behaviour of a programming language, therefore allowing for a better handling of language-specific issues.

In LP, both transformational and direct approaches have been considered. By program transformation to a TRS, a very powerful technique was obtained in [GTSKF04, SKGST06]. However, for certain language specific issues, direct approaches tend to perform better [SDS01, NDSGSK11]. This motivated the development of a combined approach in [SKGN10].

Obtaining a transformational approach for the full CHR language, successfully combined with existing techniques for LP or TRS, is hard. As was shown in [PSDS07a, PSDS07b], the subset of CHR where we allow for simplification, but not propagation, can be transformed and combined with existing approaches. For this subset, terminating programs behave similar to terminating LP programs. However, considering propagation, a transformational approach is less of an option. For single-headed propagation rules, one can still avoid the necessity of complex data structures (see [PSDS07a, PSDS07b] and Section 5.2.3). Unfortunately, this is not the case for multi-headed propagation rules.

Therefore, to handle the full CHR language, a better insight in the termination behaviour of CHR programs is required. Hence, in the remainder of this text, we will only be concerned with the development of direct approaches. Potentially, these direct approaches can then be used to build a transformational approach, allowing for reuse of existing termination provers.

1.3.2 Host language

When studying the termination problem for CHR, we have to consider the presence of a *host language*. CHR is expected to be built on top of another programming language, which, at the least, has to implement syntactic equality and the boolean relations true and false [Frü98]. Considering a host language confronts us with behaviour non-existent in other programming languages. For

one, loops in CHR may depend on the effect of executing calls to the host language. Therefore, we need to be able to estimate the effect of such calls. Furthermore, there can also be non-termination in the host language or there can be loops between CHR and the host language.

In studying the termination problem for CHR, we will focus on the behaviour of CHR programs (see Chapters 2, 3, 4 and 5). We will therefore assume universal termination of calls to the host language. Therefore, any call to the host language is assumed to return an answer in a finite number of steps, which is a non-deterministic choice between all, but finitely many, possible answers. In practice (see Chapters 6, 7, 8 and 9), we will consider termination of CHR on top of Prolog and prove termination of the Prolog part of the program as well.

Finally, we will not allow the host language to execute calls to CHR. That is, we restrict to CHR programs in which calls to the host language can only result in variable bindings and not in the addition of new CHR constraints. As such, we avoid loops between CHR and the host language.

1.3.3 Termination of simplification

A *simplification rule* in CHR represents the replacement of a multiset of CHR constraints in the constraint store by a multiset of built-in and CHR constraints. Consider for example the simplification rule

$$e(E), l(L) \Leftrightarrow l([E|L]).$$

Here, $e/1$ and $l/1$ are CHR predicates. As in Prolog syntax, we write a list of elements $e_1, \ldots, e_n$ as $[e_1, \ldots, e_n]$ and the empty list as []. We also use the Prolog style notation $[H|T]$ to represent $[H] \bowtie T$, where $\bowtie$ is concatenation.

Rule application is triggered by the fact that a multiset of constraints, present in the constraint store, matches with the head of a rule. Upon rule application, for the above rule, the two matching constraints are replaced by the constraint in the body of the rule, on which substitutions from matching are applied.

The program can easily be seen to terminate for any initial constraint store. At some point during the execution of the program, there will no longer be a multiset of constraints to fire the rule. Consider for example an initial constraint store, containing only the constraints $e(4)$ and $l([1,2])$. The above rule is applicable on this constraint store, resulting in a substitution $\sigma = \{E/4, L/[1,2]\}$ matching the head of the rule with the constraints in the constraint store. In the next constraint store, the matching constraints from the previous store are replaced by the constraint $l([E|L])\sigma = l([4,1,2])$ from

the body of the rule. Due to lack of matching constraints, the simplification rule is not applicable on the new store, and thus we reach a final state.

In general, direct approaches construct a termination proof on the basis of a mapping of program states to some *well-founded* set. If the mapping is such that a decrease can be shown for the *state-sizes* of consecutive program states, then, by well-foundedness of the set, we prove termination of the program. For the program above, counting the number of constraints in the constraint store is a mapping of states to the well-founded domain of positive integer numbers. Since for any application of the simplification rule the number of constraints in the constraint store strictly decreases, we prove termination.

The example above suggests that proving state-size decreases, like in LP, is useful to prove termination of a CHR program. In particular, as shown in [Frü00], this is so when we only allow for simplification in CHR programs. In Chapter 2, we discuss this in further detail using [Frü00] as a basis.

1.3.4 The effect of matching

Rule application in CHR involves *matching*, while in LP it involves unification. Consider for example the Prolog clause

$$a(s(N)) :- a(N).$$

Here, $a/1$ is a Prolog predicate, $s/1$ a functor and N a variable. Resolving a query $a(N)$ with a renamed variant of the Prolog clause from above, results in a substitution $\{N/s(N')\}$, and thus a next query $a(N')$. This query can be resolved in the same way as before. Thus, we get non-termination.

Consider now a similar CHR rule

$$a(s(N)) \Leftrightarrow a(N).$$

Here, $a/1$ is a CHR predicate, $s/1$ a functor and N a variable. For a constraint store containing only the constraint $a(N)$, the simplification rule from above is not applicable. For a renamed variant of the rule, there is no substitution for the head $a(s(N'))$ to match the constraint $a(N)$. Thus, we get termination.

Note however that unification can still be provided through the host language and, therefore, can result in bindings that are propagated backward. So, we may get non-termination similar to the LP-clause from above.

Consider for example the simplification rule

$$a(N) \Leftrightarrow N = s(M), a(M).$$

Here, $=/2$ is the built-in unification of Prolog. A renamed variant of the simplification rule is applicable on a constraint $a(N)$ in the constraint store, resulting in a substitution $\sigma = \{N'/N\}$. As a result, $a(N)$ is removed and the built-in constraint $N = s(M')$ and CHR constraint $a(M')$ are added to the store. As $N = s(M')$ will not result in failure of the program and $a(M')$ can fire the rule in the same way as before, we get non-termination.

1.3.5 The effect of guarded rules

Until now, we only considered CHR programs containing CHR rules without a *guard*. However, in practice, most CHR rules have a guard. In CHR, guards can only be composed of built-in constraints, which —for the rule to be applicable— need to be satisfiable given the substitutions resulting from matching.

The following example, where Prolog is used as a host language for CHR, illustrates the use of guards in CHR.

Example 1.3.1 (introductory example)**.** *Consider the CHR(Prolog) program*

R_1 @ $gcd(0) \Leftrightarrow true$.
R_2 @ $gcd(M), gcd(N) \Leftrightarrow N >= M, M > 0 \mid L \text{ is } N - M, gcd(M), gcd(L)$.

The program computes the greatest common divisor by the Euclidean algorithm. An initial constraint store for the program is any number of $gcd/1$ *constraints with integers greater than or equal to* 2*. Consider for instance an initial constraint store of two constraints* $gcd(4)$ *and* $gcd(2)$*. The second rule is applicable on this constraint store. The guard of the rule guarantees that* $gcd(2)$ *will match the first head,* $gcd(M)$*, and* $gcd(4)$ *the second head,* $gcd(N)$*, of the rule. When the rule is applied,* $gcd(4)$ *and* $gcd(2)$ *are replaced with a fresh* $gcd(2)$*,* L' *is* $4 - 2$ *and* $gcd(L')$ *constraint. As* $gcd(L')$ *is uninstantiated, it cannot be used to fire the second rule. For* $gcd(L')$*, the guard of* R_2 *cannot be satisfied. Therefore, first the built-in constraint* L' *is* $4 - 2$ *must be solved, instantiating* L' *to* 2*.*

Next, the two $gcd(2)$ *constraints of the constraint store are replaced with a fresh* $gcd(2)$*,* L'' *is* $2-2$ *and* $gcd(L'')$ *constraint. Solving the built-in constraint yields a constraint store of one* $gcd(2)$ *and one* $gcd(0)$ *constraint, the latter of which is removed by the first rule. As no more rules are applicable on the resulting constraint store, containing only* $gcd(2)$*, the computation terminates.* □

For termination, it is important to consider that guards result in variable bindings. Consider a similar rule as in the previous subsection:

$$a(N) \Leftrightarrow N = s(M) \mid a(M).$$

One would be inclined to think that since the unification is now part of the guard, that its only purpose is to be checked for satisfiability, unable to result in any variable bindings. This, however, is not the case. A CHR guard, when checked for satisfiability, generates the bindings for which the guard became satisfied. Therefore, the guard can bind variables in the body and the head of a rule. For the rule considered above this means that when we apply a renamed variant of the rule on a CHR constraint $a(N)$, it first results in a matching substitution $\sigma = \{N'/N\}$ and then in an answer substitution $\theta = \{N/s(M')\}$ for satisfaction of the guard. As the constraint $a(M')\sigma\theta$ replacing $a(N)$ can fire the rule in the same way as before, we get non-termination.

Finally, it is important to consider that bindings are applied to the entire constraint store and thus can result in further instantiations of constraints not involved in the application of some rule.

1.3.6 Termination of propagation

The presence of propagation makes the termination problem of CHR hard. Consider for example the *propagation rule*

$$a(s(N)) \Rightarrow a(N).$$

Given a CHR constraint $a(s(s(0)))$ in the constraint store, the rule adds a CHR constraint $a(s(0))$, without removing the constraint $a(s(s(0)))$ used to fire the rule. Since propagation rules do not remove the constraints on which they fire, they are infinitely applicable. Therefore, to avoid the *trivial non-termination* of propagation, propagation rules must respect a *fire-once policy*: they can only be activated *once* on any matching multiset of constraints. Under this restriction and taking into account that CHR uses matching instead of unification, a program consisting only of the above rule is terminating.

Now, consider a slight variant of the above propagation rule:

$$a(s(N)), a(N) \Rightarrow a(N).$$

Based on LP-termination intuition, we would think that this rule is terminating as well. We have only strengthened the precondition of the rule and there is still a decrease in the overall "size" between the heads and the body of the rule. The latter rule, however, is non-terminating for any constraint store on which it can be activated at least once. The reason is that each time the rule is activated, say on a constraint store containing the constraints $a(s(s(0)))$ and $a(s(0))$, it adds a new constraint $a(s(0))$. This "fresh" copy of $a(s(0))$, recombined with $a(s(s(0)))$, can fire the rule again. Thus, we get non-termination.

As mentioned earlier, in LP termination analysis, one important intuition is to map states in the computation to elements of a well-founded set, such that consecutive states correspond to decreasing elements in the well-founded set. Thus, an interesting question is whether we could find a representation of a "state" in CHR computations that, by means of a mapping to a well-founded set, can be seen to decrease for terminating CHR programs with propagation.

It turns out that this is not an easy problem. Even for the terminating CHR program consisting only of the propagation rule $(a(s(N)) \Rightarrow a(N).)$ from above, the constraint store increases in size after each rule application: for instance a constraint store containing only the constraint $a(s(s(0)))$ is changed into a constraint store containing the constraints $a(s(s(0)))$ and $a(s(0))$. Thus, the constraint store itself is not sufficient as such a representation of a state.

For this reason, in [VDSP08] (see Chapter 3), a new approach to termination analysis of CHR was developed. Although the approach can prove termination of CHR programs with propagation, it is not based on the intuition of state-size decreases. In [PDS08a] (see Chapter 4), an answer is provided to the question above, introducing a new representation of states in CHR computations that can be seen to decrease in size, even for terminating CHR programs with propagation rules. As it turned out, this approach is strictly more powerful than both the approaches of [Frü00] and [VDSP08] (see Section 5.1).

A final difficulty, related to propagation, is the presence of *simpagation rules* in CHR programs. This third kind of CHR rules triggers also on constraints in the constraint store which are not removed by the rule, resulting in unexpected program behaviour. Consider for example the CHR program

$$a(A) \setminus b(A) \Leftrightarrow true.$$

The CHR rule, a simpagation rule, differs only from a simplification rule in that the head is split into a *kept head*, left of the backslash, and a *removed head*, right of it. Without propagation, the rule can be replaced by the simplification rule $(a(A), b(A) \Leftrightarrow a(A).)$. However, in the presence of propagation, $(a(A) \setminus b(A) \Leftrightarrow true.)$ may not be equivalent to $(a(A), b(A) \Leftrightarrow a(A).)$, since the latter introduces a "fresh" copy of $a(A)$ that again can fire some propagation rule.

1.3.7 Availability of different control structures

Many different forms of control for CHR programs were considered over the years [SVWSDK10]. In its most basic form, CHR control is defined by the *abstract CHR semantics*. Most closely related to the declarative semantics underlying CHR, the abstract semantics does not consider a fire-once policy, and thus, in the presence of propagation, results in non-terminating programs.

For reasons of practicality the *theoretical CHR semantics* was conceived, implementing a fire-once policy. Operationally, two equivalent but different approaches were pursued. On the one hand, as available in most implementations of CHR [Sch05], a *propagation history* can be used to prevent multiple applications of a propagation rule on the same combination of constraints. On the other hand, by a *token store* (or *propagation forecast*), propagation rules can be granted the right to fire on combinations of constraints [Abd97].

According to the abstract and theoretical CHR semantics, rules are applied until exhaustion. Which rules are applied is a non-deterministic choice between all possible applications for a given constraint store. Rule application is a committed choice and thus cannot be undone, unlike in LP where there is search. Hence, the concepts of *universal* and *existential termination* from LP do not apply to the CHR context. However, since in general we need to consider all possible execution strategies to prove termination of a CHR program, CHR-termination is most closely related to universal termination in LP.

To obtain more efficient program execution, other CHR semantics [DSGH04, DK08, SVWSDK10] were conceived. These are in general instances of the theoretical CHR semantics. Due to being more specific, certain CHR programs which terminate when executed under a more refined semantics are non-terminating for the theoretical CHR semantics. Nevertheless, for most CHR programs the termination properties turn out to be the same. Therefore, in the next chapters, we work towards a theory for proving termination of CHR programs executed under the theoretical CHR semantics and refer to Section 5.2.4 for a discussion on termination of CHR programs for more refined control.

1.4 Overview

Because the termination (halting) problem is undecidable for Turing complete programming languages like CHR [Sne08], the past research on termination analysis proceeded mainly along the following two lines [DSD94]:

- Provide theoretical conditions, (possibly necessary and) sufficient for termination. These conditions provide a better theoretical understanding of the termination problem, but cannot be verified automatically. In LP, significant effort has been put in finding such conditions (see e.g., [AP91, DSD94, DSS02]).
- Provide verifiable conditions, sufficient for termination. These conditions serve as a basis for verification tools and are derived on the basis of theoretical termination conditions (see e.g., [DDSV99, GC05, SDS03,

GTSKF04, SDS04, SDS05b, SDS05a, MB05, BCG+07, GSKT+07, EWZ08, NDSGSK11, SKGN10]).

Until recently, there was only an informal discussion on termination of CHR without propagation [Frü00]. In the *first part* of the thesis we are therefore concerned with the development of a theoretical framework for termination analysis of the *full* CHR language. First, in Chapter 2, we introduce a formal framework for termination analysis of CHR without propagation, extending the approach of [Frü00] to general orderings. Then, in Chapter 3, we discuss the first approach to termination analysis of CHR programs with propagation [VDSP08] based on a new concept of termination: the *finite addition of constraints*; and extend it to general orderings as well. In Chapter 4, we generalise the approaches of Chapters 2 and 3 [PDS08a], resulting in the most powerful theoretical framework to handle termination of the full CHR language, yet. This approach is based on a new state representation for CHR that can be seen to decrease for terminating CHR programs. Finally, in Chapter 5, we conclude the first part of the thesis on theoretical frameworks for termination analysis of CHR.

The *second part* of the thesis is dedicated to automating termination proofs for CHR. First, in Chapter 6, we adapt the constraint-based approach of [NDSGSK11] with polynomial interpretations to the CHR context, extending it to integer polynomial interpretations. As such, we obtain an integrated constraint-based approach, strictly more powerful than the approach of [NDSGSK11], which is also capable of proving termination of bounded increase problems and problems that only terminate for specific kinds of queries. Then, in Chapter 7, we discuss modular approaches to termination analysis of CHR and introduce the notion of a *CHR cycle*, which differs significantly from the notion with the same name in LP. In Chapter 8, we discuss and evaluate *T*CoP*, a termination analyser for CHR on top of Prolog, based on the techniques of Chapters 6 and 7. Then, in Chapter 9, we conclude the second part of the thesis on automating termination proofs for CHR.

Finally, in Chapter 10, we conclude the thesis: *Termination analysis of CHR: Theory and practice*.

Part I

Theory

Chapter 2

Termination of abstract CHR

The first work on termination of CHR programs [Frü00] was for several years the only work on the topic. By relating LP-termination to CHR-termination, an approach for proving termination of CHR solvers, containing only simplification, was developed. The approach, however, cannot easily be extended to prove termination of CHR solvers with propagation.

The proof method —underlying many of the approaches in LP-termination as well— is based on proving *state-size decreases on consecutive states* by means of a *well-founded ordering on program states*. In [Frü00], such a well-founded ordering on states is defined on the basis of the constraint store. It is therefore obvious that propagation, which can only result in explicit size-increases of the constraint store, cannot be handled by this technique.

Even though the approach discussed in [Frü00] is not applicable to CHR programs with propagation, it is a useful and interesting approach to consider. In particular, it is interesting for determining the extent to which concepts in LP-termination relate to concepts in CHR-termination, without having to be concerned with language specifics such as a fire-once policy. This knowledge can afterwards be used in the development of new approaches which are applicable to the full setting of the CHR language.

For this reason, we focus in this chapter on termination of CHR under the abstract CHR semantics, using [Frü00] as a basis. Note however that we extend the approach of [Frü00] to general orderings. Finally, note that for CHR programs without propagation, termination under the abstract semantics

is equivalent to termination under the theoretical semantics [Frü98].

2.1 CHR syntax

We assume familiarity with LP and its main results [Llo87, Apt90]. By $\mathcal{L}$, we denote the first order language underlying a CHR program P. By Var_P, $Const_P$, Fun_P and $Pred_P$, we denote, respectively, the sets of variables, constants, function and predicate symbols of $\mathcal{L}$. By U_P and B_P, we denote, respectively, the Herbrand universe and Herbrand base. These are the sets of ground terms and atoms constructible from $\mathcal{L}$. Finally, by $Term_P$ and $Atom_P$, we denote, respectively, the sets of all terms and atoms constructible from $\mathcal{L}$.

Similar to LP, the rules of a CHR program operate on first order predicates p/n, with $n \geq 0$, where p is a predicate symbol and n the predicate's *arity*. In CHR, we call atoms based on these symbols *constraints* [Frü98]. Given a constraint $c(t_1, \ldots, t_n)$, we denote by $rel(c(t_1, \ldots, t_n))$ its predicate c/n.

Definition 2.1.1 (constraint)**.** Constraints $c(t_1, \ldots, t_n)$*, with* $n \geq 0$*, are first-order predicates applied to terms* t_i*, with* $1 \leq i \leq n$*. We distinguish between two kinds of constraints:* CHR constraints *are user-defined and solved by the* CHR rules *of a* CHR program*;* built-in constraints *are pre-defined and solved by a* constraint theory *(*CT*) defined in the host language.* □

Constraints are kept in a constraint store, which can be represented as a *multiset* (or bag) of constraints. Therefore, to formalise the syntax (and semantics) of CHR, we first recall some multiset theory [Bli89, DM79].

Definition 2.1.2 (multiset)**.** *A* multiset *is a tuple* $M_S = \langle S, m_S \rangle$*, where* S *is a set, called the* underlying set *of elements, and* m_S *a* multiplicity function*, mapping the elements* e *of* S *to strictly positive integer numbers* $m_S(e) \in \mathbb{N}_0$ *that represent the number of occurrences of* e *in* M_S*.* □

Like any function, m_S may be represented as a set $\{(s, m_S(s)) \mid s \in S\}$ of ordered pairs.

Example 2.1.1 (multiset)**.** *Examples of multisets are* $\langle \{a, b\}, \{(a, 2), (b, 1)\} \rangle$ *and* $\langle \{a, b\}, \{(a, 1), (b, 1)\} \rangle$*. We introduce the alternative notation to represent these multisets as* $[\![a, a, b]\!]$ *and* $[\![a, b]\!]$*, respectively.* □

If a *universe* U in which the elements of S must live —not to be confused with the Herbrand universe U_P— is specified, the definition of a multiset can be simplified to a function $\mu_S : U \to \mathbb{N}$, from U to the natural numbers, obtained by extending m_S to U with values 0 outside S.

Definition 2.1.3 (multiset indicator function)**.** *Let U be the universe of a multiset $M_S = \langle S, m_S \rangle$. Then, we define the* multiset indicator function *$\mu_S : U \to \mathbb{N}$, of M_S w.r.t. U, as:*

$$\begin{cases} \mu_S(u) = m_S(u) & \text{if } u \in S, \\ \mu_S(u) = 0 & \text{if } u \notin S. \end{cases}$$ □

Note that $Atom_P$ characterises a universe for the constraint stores turning up in computations of an abstract CHR program P. Since it is often more favourable to discuss multisets in terms of some universe, we can —as an abuse of notation— denote a multiset $M_S = \langle S, m_S \rangle$ also by $\langle U, \mu_S \rangle$, or simply by μ_S if the universe is clear from context. Note that the distinction is made by the symbol m for multiplicity functions and μ for multiset indicator functions, possibly with sub -or superscripts.

Furthermore, we will need the notion of a *multisubset* in the constraint store since the rules of a CHR program will be applied on multisubsets of constraints.

Definition 2.1.4 (multisubset)**.** *Let U be a universe for the multisets M_{S_1} and M_{S_2}. Then, multiset M_{S_1} is a* multisubset *of multiset M_{S_2}, denoted $M_{S_1} \sqsubseteq M_{S_2}$, iff the multiset indicator functions μ_{S_1} and μ_{S_2} w.r.t. U, of M_{S_1} and M_{S_2} respectively, are such that $\mu_{S_1}(u) \leq \mu_{S_2}(u)$ for all u in U. Associated to $\sqsubseteq$, we define* strict multisubsets*: $M_{S_1} \sqsubset M_{S_2} \leftrightarrow M_{S_1} \sqsubseteq M_{S_2} \wedge M_{S_2} \not\sqsubseteq M_{S_1}$.* □

Note that $M_{S_1} = M_{S_2} \leftrightarrow M_{S_1} \sqsubseteq M_{S_2} \wedge M_{S_2} \sqsubseteq M_{S_1}$.

Example 2.1.2 (multisubset)**.** *Consider two multisets $A = [\![1, 1, 2]\!]$ and $B = [\![1, 1, 2, 2, 3]\!]$ with universe $U = \{1, 2, 3\}$. Then, $A \sqsubseteq B$ because $\mu_A(1) \leq \mu_B(1)$, $\mu_A(2) \leq \mu_B(2)$ and $\mu_A(3) \leq \mu_B(3)$. Furthermore, because $B \not\sqsubseteq A$, $A \sqsubset B$.* □

Finally, we will need the *join* operator, for adding multisets together.

Definition 2.1.5 (multiset join)**.** *Let M_A and M_B be multisets with universe U and let u be an element of U. Then, the multiset indicator function of the* join *$M_C = M_A \uplus M_B$ is given by the sum $\mu_C(u) = \mu_A(u) + \mu_B(u)$ of the multiset indicator functions of M_A and M_B.* □

Example 2.1.3 (multiset join)**.** *Consider two multisets $A = [\![1, 1, 2]\!]$ and $B = [\![1, 1, 2, 2, 3]\!]$ with universe $U = \{1, 2, 3\}$. Then, $A \uplus B = C = [\![1, 1, 1, 1, 2, 2, 2, 3]\!]$. Obviously, $A \sqsubseteq C$ and $B \sqsubseteq C$.* □

We are now ready to define the syntax of the three kinds of *CHR rules* occurring in *CHR programs* [Frü98].

Definition 2.1.6 (CHR program)**.** *A* CHR program *is a finite set of* CHR rules*. Three kinds of CHR rules can be present in a CHR program:*

- *A* simplification rule

$$H_1, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

  is applicable on a multiset of CHR constraints in the constraint store, matching with the head $[\![H_1, \ldots, H_n]\!]$ *of the simplification rule, such that the* guard $[\![G_1, \ldots, G_k]\!]$ *is satisfiable. Upon application, the multiset of constraints matching with the head is replaced by an appropriate instance of the multiset of built-in and CHR constraints* $[\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]$ *from the* body *of the rule.*

- *A* propagation rule

$$H_1, \ldots, H_n \Rightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

  is applicable on a multiset of CHR constraints in the constraint store, matching with the head $[\![H_1, \ldots, H_n]\!]$ *of the rule, such that the guard* $[\![G_1, \ldots, G_k]\!]$ *is satisfiable. Upon application, an appropriate instance of the multiset of built-in and CHR constraints* $[\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]$ *from the body of the rule is added to the constraint store, while none of the constraints used for rule application are removed.*

- *Finally, a* simpagation rule

$$H_1, \ldots, H_j \setminus H_{j+1}, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

  is applicable on a multiset of CHR constraints in the constraint store, matching with the head $[\![H_1, \ldots, H_n]\!]$ *of the rule, such that the guard* $[\![G_1, \ldots, G_k]\!]$ *is satisfiable. Upon application, the part, matching with the* removed head $[\![H_{j+1}, \ldots, H_n]\!]$ *is replaced by an appropriate instance of the multiset of built-in and CHR constraints* $[\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]$ *from the body of the rule, while none of the constraints matching with the* kept head $[\![H_1, \ldots, H_j]\!]$ *are removed.*

Note that the guards $[\![G_1, \ldots, G_k]\!]$ *of CHR rules may only consist of built-in constraints. Also note that a CHR rule can optionally be named by adding "rulename @ " in front of the rule.* □

Even though the constraint store can be represented as a multiset, it corresponds to a conjunction of constraints. CHR rules thus rewrite conjunctions of constraints. Therefore, as the syntax suggests, simplification

(or simpagation) corresponds *declaratively* to (a conditional) logical equivalence while propagation corresponds to logical implication.

The next example program —an introductory example to CHR(Prolog)— does not contain propagation as it is not essential to discuss propagation until the next section. An example program with propagation is more appropriate there (see Example 3.1.1).

Example 2.1.4 (CHR(Prolog) syntax)**.** *The program below is the merge-sort algorithm in CHR(Prolog). The initial constraint store consisting of the CHR constraint* $msort(L)$*, with* L *a list of natural numbers of length exactly* 2^n*, yields a tree-representation of the order, which then is rewritten into a sorted list of elements.*

$$
\begin{aligned}
&R_1 \ @\ msort([]) \Leftrightarrow true.\\
&R_2 \ @\ msort([L|Ls]) \Leftrightarrow r(0,L), msort(Ls).\\
&R_3 \ @\ r(D,L1), r(D,L2) \Leftrightarrow less(L1,L2) \mid r(s(D),L1), a(L1,L2).\\
&R_4 \ @\ a(L1,L2) \setminus a(L1,L3) \Leftrightarrow less(L2,L3) \mid a(L2,L3).
\end{aligned}
$$

The first two rules decompose a list of elements, while adding new $r/2$ *constraints to the store. The constraints* $r(D,L)$ *represent trees of depth* D *(initially 0) and root value* L*. The third and fourth rule perform the actual merge-sorting. The third rule joins two trees of equal depth. It replaces both trees by a new tree of incremented depth, where —as represented by* $a/2$ *constraints— the largest root becomes a child node of the smallest, hence the branch is ordered. Note that the initial list needs to be a set and not a multiset of natural numbers for correct behaviour and that its length has to be a power of 2 to guarantee that one ends with a single tree. Finally, the fourth rule merge-sorts different branches of a tree into a single branch, i.e., an ordered list of elements.* □

2.2 The abstract CHR semantics

The operational semantics of a CHR program is defined as a state transition system [Frü98]. In its simplest form, called the *abstract CHR semantics*, it defines a *CHR state* as a multiset of constraints.

Definition 2.2.1 (abstract CHR state)**.** *An* abstract CHR state *is a multiset* Q *of constraints, with universe* $Atom_P$*, called a* constraint store. □

The *CHR transition relation* represents the consecutive CHR states for a given CHR program P and constraint theory CT. As the abstract CHR semantics

does not define a fire-once policy for propagation rules, we can, without loss of generality, represent the three kinds of rules as simplification rules only.

That is, a propagation rule

$$H_1, \ldots, H_n \Rightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

can be written as a simplification rule

$$H_1, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, H_1, \ldots, H_n, C_1, \ldots, C_m.$$

and a simpagation rule

$$H_1, \ldots, H_j \setminus H_{j+1}, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

as a simplification rule

$$H_1, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, H_1, \ldots, H_j, C_1, \ldots, C_m.$$

The transition relation for CHR rules, under the abstract CHR semantics, is therefore only defined in terms of simplification.

Definition 2.2.2 (abstract CHR transition relation)**.** *Let P be a CHR program and CT a constraint theory for the built-in constraints. Let θ represent the substitutions corresponding to the bindings generated when resolving built-in constraints by CT and let σ represent substitutions for the variables in the head of the rule as a result of matching. Then, the* transition relation $\rightarrow_P$ *for P, where Q is a CHR state, is defined by:*

1. ***A solve transition**:*
   ***if** $Q = [\![b]\!] \uplus Q'$, with b a built-in constraint, and $CT \models b\theta$,*
   ***then** $Q \rightarrow_P Q'\theta$,*
   ***else if** $Q = [\![b]\!] \uplus Q'$, $b \neq false$ and $\forall \theta : CT \not\models b\theta$,*
   ***or** $Q = [\![b]\!] \uplus Q'$, $b = false$ and $Q' \neq \emptyset$,*
   ***then** $Q \rightarrow_P [\![false]\!]$.*
2. ***Simplification**:*
   ***if** $(H_1, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.)$ in P,*
   ***and if** $Q = [\![h_1, \ldots, h_n]\!] \uplus Q'$ and*
   $CT \models (h_1 = H_1\sigma) \wedge \cdots \wedge (h_n = H_n\sigma) \wedge (G_1 \wedge \cdots \wedge G_k)\sigma\theta$,
   ***then** $Q \rightarrow_P ([\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!] \uplus Q')\sigma\theta$.*

Rules in CHR are non-deterministically applied until exhaustion, thus until no more transitions are possible. Which rule is applied is a committed choice and thus cannot be undone. Built-in constraints are assumed to return an answer in a finite number of steps. Solving built-ins can therefore only result in variable bindings. If built-in constraints cannot be solved by the CT, the CHR program fails, returning a CHR state $[\![false]\!]$. □

As can be noticed, the bindings resulting from guard satisfaction are considered. Therefore, even though it is considered a bad practice, a CHR programmer can use the guard to bind variables in the head of a rule. Thus, as in LP, we need to consider backpropagation of data.

For the abstract CHR semantics, we define the following kinds of CHR states.

Definition 2.2.3 (abstract initial and final CHR states)**.** *Let P be a CHR program with transition relation $\rightarrow_P$. An* abstract initial CHR state *or* query state *is an abstract CHR state. An* abstract final CHR state *or* answer state *is a state Q, such that no CHR state Q' exists for which $(Q, Q') \in \rightarrow_P$.*

A final CHR state Q is a failed CHR state *if $Q = [\![false]\!]$, otherwise Q is a* successful CHR state*.* □

We demonstrate the abstract CHR semantics in the next example, where we discuss an execution trace for the Merge-sort program from Example 2.1.4.

Example 2.2.1 (CHR semantics)**.** *Executing the Merge-sort program on an initial CHR state*

$$Q_0 = [\![msort([s(s(s(0))), 0, s(s(0)), s(0)])]\!],$$

where we represent natural numbers by means of a successor notation*:* $0, s(0), s(s(0)), \ldots$*; yields a next state*

$$Q_1 = [\![r(\underline{0}, s(s(s(0)))), msort([0, s(s(0)), s(0)])]\!].$$

Note that we underline the first argument of $r/2$ constraints for readability. As R_2 remains to be the only rule applicable, from Q_1, we obtain the state

$$Q_2 = [\![r(\underline{0}, s(s(s(0)))), r(\underline{0}, 0), msort([s(s(0)), s(0)])]\!].$$

On Q_2, both R_2 and R_3 are applicable. Which rule is selected is a non-deterministic choice. Thus, one possibility is to keep applying R_2, leading to

$$Q_4 = [\![r(\underline{0}, s(s(s(0)))), r(\underline{0}, 0), r(\underline{0}, s(s(0))), r(\underline{0}, s(0)), msort([])]\!].$$

On Q_4, R_2 is no longer applicable, but R_3 is still applicable and R_1 became applicable. We apply R_1 and thus replace $msort([])$ by the trivially succeeding built-in constraint true (removed by default):

$$Q_5 = [\![r(\underline{0}, s(s(s(0)))), r(\underline{0}, 0), r(\underline{0}, s(s(0))), r(\underline{0}, s(0))]\!].$$

Only R_3 is applicable on Q_5, however, can fire in six different ways. We select $r(\underline{0}, 0)$ and $r(\underline{0}, s(s(0)))$ to match with the first and second head of R_3, respectively. This results in a matching substitution $\sigma = \{D/0, L1/0, L2/s(s(0))\}$, for which the call $less(L1, L2)\sigma$ to Prolog succeeds. We obtain

$$Q_6 = [\![r(\underline{0}, s(s(s(0)))), r(\underline{0}, s(0)), r(\underline{s(0)}, 0), a(0, s(s(0)))]\!].$$

We fire R_3 again, with the constraints $r(\underline{0}, s(0))$ and $r(\underline{0}, s(s(s(0))))$, and get

$$Q_7 = [\![r(\underline{s(0)}, s(0)), r(\underline{s(0)}, 0), a(0, s(s(0))), a(s(0), s(s(s(0))))]\!].$$

Now R_4 becomes applicable. We continue however to apply R_3 and obtain

$$Q_8 = [\![r(\underline{s(s(0))}, 0), a(0, s(s(0))), a(s(0), s(s(s(0)))), a(0, s(0))]\!]$$

on which R_3 is no longer applicable.

Again, several combinations of constraints in the constraint store can fire R_4. By matching $a(0, s(0))$ to the kept head of R_4 and $a(0, s(s(0)))$ to the removed head of R_4, we get the state

$$Q_9 = [\![r(\underline{s(s(0))}, 0), a(0, s(0)), a(s(0), s(s(0))), a(s(0), s(s(s(0))))]\!].$$

Finally, by matching $a(s(0), s(s(0)))$ and $a(s(0), s(s(s(0))))$ with the kept and removed head, respectively, we obtain the state

$$Q_{10} = [\![r(\underline{s(s(0))}, 0), a(0, s(0)), a(s(0), s(s(0))), a(s(s(0)), s(s(s(0))))]\!].$$

There are no more rules applicable on Q_{10} and thus we reach an answer state, representing an ordered list of elements as a graph, with root node $r/2$ and directed arcs $a/2$. Obviously, Q_{10} is a successful CHR state. □

2.3 Termination of abstract CHR

The termination behaviour of an abstract CHR program is very similar to that of an LP program. Discussing termination of the abstract CHR semantics is therefore mainly concerned with the introduction of several of the concepts from LP-termination [DSD94].

2.3.1 Termination of abstract CHR

First, we define what it means for a CHR program to be terminating. By termination, we understand that a program, given its *intended use*, should return an answer state in a finite number of steps. By considering a program's intended use, more precise results can be obtained, as programs may be non-terminating when used improperly.

Definition 2.3.1 (CHR termination)**.** *A CHR program P with transition relation $\rightarrow_P$ is* terminating *for an initial CHR state Q_0 iff P returns an answer state for Q_0 using transitions in $\rightarrow_P$.* □

Since there is full non-determinism in the application of CHR rules, we need to consider all possible execution strategies. Considering this, we can formulate a more useful notion of CHR-termination on the basis of *CHR computations*.

Definition 2.3.2 (CHR computation)**.** *Let P be a CHR program and $\rightarrow_P$ its transition relation. We say that a transition (Q, Q') and a transition (Q'', Q''') are* connected *iff the CHR states Q' and Q'' are equal. Then, a* computation *of P for an initial CHR state Q_0, is a possibly infinite sequence (or chain)*

$$(Q_0, Q_1), (Q_1, Q_2), \ldots, (Q_{n-1}, Q_n), \ldots$$

of transitions in $\rightarrow_P$, where each of the consecutive transitions in the sequence are connected, such that the sequence terminates only if it cannot be extended further with transitions in $\rightarrow_P$. A finite computation $(Q_0, Q_1), (Q_1, Q_2), \ldots, (Q_{n-1}, Q_n)$ is a failed computation *if Q_n is a failed state. Otherwise, a finite computation is a* successful computation*.* □

We use a more intuitive notation for computations, i.e., instead of writing $(Q_0, Q_1), (Q_1, Q_2), \ldots$, we write

$$Q_0 \xrightarrow{P} Q_1 \xrightarrow{P} Q_2 \xrightarrow{P} \ldots.$$

CHR computations represent execution traces of a CHR program. If for a given initial CHR state all possible computations are finite, then an answer state is returned. Therefore, we can formulate the following lemma.

Lemma 2.3.1 (CHR termination)**.** *A CHR program P with transition relation $\rightarrow_P$ is terminating for an initial CHR state Q_0 iff all computations of P for Q_0 are finite.* □

Proof. Trivial. □

Note that a program is said to be terminating for a set of initial CHR states if the program terminates for each initial CHR state in the set. Furthermore, note that Definition 2.3.1, Definition 2.3.2 and Lemma 2.3.1 are not defined in terms of a CHR semantics, but in terms of a CHR transition relation. Thus, they are applicable to any CHR semantics defined by a transition relation.

2.3.2 A termination proof by a well-founded order

In general, in LP, termination is proven on the basis of derivations. A proof of termination is established if a well-founded ordering on states can be defined for which consecutive program states demonstrate size-decreases. A similar strategy can be applied to CHR as well, using computations. As such, we can guarantee finiteness of all computations and thus termination. We first recall some basic concepts of orderings [DM79, Der82, SDS00].

Definition 2.3.3 (pre-order)**.** *Let S be a set and $\succeq$ a binary relation on S. Then, $\succeq$ is a* pre-order *or* quasi-order *iff it is* reflexive *and* transitive*. That is, for all a, b and c in S we have that:*

- *$a \succeq a$ (reflexivity),*
- *and if $a \succeq b$ and $b \succeq c$ then $a \succeq c$ (transitivity).*

If $\succeq$ is furthermore antisymmetric*, i.e., if $a \succeq b$ and $b \succeq a$ then $a = b$, it is a* partial order*. On the other hand if it is* symmetric*, i.e., if $a \succeq b$ then $b \succeq a$, it is an* equivalence relation*.* □

Every binary relation R on a set S can be extended to a pre-order on S, this by taking the transitive and reflexive closure $R^{+=}$. Given a pre-order $\succeq$ on S, an equivalence relation $\approx$ on S can be defined as $a \approx b$ iff $a \succeq b$ and $b \succeq a$. Note that $\approx$ is reflexive since a pre-order is reflexive, transitive, by applying transitivity of the pre-order twice, and symmetric by definition. Using this relation, it is possible to construct a partial order on the quotient set $S/\approx$ of the equivalence, i.e., the set of all equivalence classes of $\approx$. By the construction of $\approx$, the partial order is independent of the chosen representatives of these equivalence classes, hence the corresponding relation is well-defined.

Example 2.3.1 (orderings)**.** *By checking their properties, one immediately sees that the well-known orders on natural numbers, integers, rational numbers and reals are all orders in the above sense. However, they have the additional property of being* total*, i.e., for all a and b in S: $a \succeq b$ or $b \succeq a$ (totality).* □

For a pre-order $\succeq$, a relation $\succ$ can be defined as $a \succ b$ iff $a \succeq b$ and $b \not\succeq a$. It is a *strict partial order*, and every strict partial order can be the result of such a construction. Therefore, we refer to it as the strict partial order $\succ$ *associated* to a pre-order $\succeq$. If the pre-order is anti-symmetric, hence a partial order $\succeq$, the equivalence is equality.

Finally, interesting for termination analysis is *well-foundedness* of a strict partial ordering.

Definition 2.3.4 (well-foundedness)**.** *A strict partial ordering on S is* well-founded *iff no infinite strict decreasing chain, $s_1 \succ s_2 \succ \cdots \succ s_n \succ \ldots$, of elements in S exists.* □

Example 2.3.2 (well-foundedness)**.** *Consider a binary relation $\div_{\mathbb{N}}$ on natural numbers, given by dividability. Then, for example $(9, 3)$ is in the relation. As can be verified, the relation $\div_{\mathbb{N}}$ is a pre-order on natural numbers. More specifically, it is a non-total partial order. Thus, not all natural numbers are comparable: e.g., $(9, 4)$ is not in $\div_{\mathbb{N}}$. As furthermore can be verified, the strict partial order associated to $\div_{\mathbb{N}}$ is well-founded on $\mathbb{N}$.*

Similarly, a binary relation $\div_{\mathbb{Z}}$ on integer numbers can be defined. This relation is a pre-order with an associated strict partial order that is well-founded on $\mathbb{Z}$. However, in contrast to $\div_{\mathbb{N}}$, it is not a partial order: e.g., $(3, -3)$ and $(-3, 3)$ are in $\div_{\mathbb{Z}}$. □

To prove termination of a CHR program P for a query Q_0, we show size-decreases between consecutive program states. In order to show this, we need to define a pre-order on CHR states with an associated strict partial ordering that is well-founded. We refer to this construction as a *reduction pair*, a notion similar to the reduction pairs of [NGSKDS08], however, defined on the basis of associated orderings [DSS02] and not compatible orderings [NDSGSK11].

Definition 2.3.5 (reduction pair)**.** *Let S be a set, $\succeq$ a pre-order relation on S and $\succ$ the strict partial order relation associated to $\succeq$. Then, $(\succeq, \succ)$ is a* reduction pair *for S iff $\succ$ is well-founded on S.* □

Given a reduction pair $(\succeq, \succ)$ on CHR states such that in every computation

$$Q_0 \xrightarrow{P} Q_1 \xrightarrow{P} Q_2 \xrightarrow{P} \ldots \xrightarrow{P} Q_{n-1} \xrightarrow{P} Q_n,$$

with initial CHR state Q_0, the consecutive states, $(Q_{i-1}, Q_i) \in \rightarrow_P$, demonstrate a strict decrease $Q_{i-1} \succ Q_i$, then all computations must be finite by well-foundedness of $\succ$. Thus, by Lemma 2.3.1, P is terminating for Q_0.

Example 2.3.3 (reduction pair)**.** *Consider a CHR program P, consisting of the rule $(R \ @ \ a, a \Leftrightarrow a, b.)$. Then, if we define a reduction pair $(\succeq, \succ)$ on CHR states on the basis of the multiplicity of $a/0$ constraints in the state, we can prove that a strict decrease exists between any two consecutive states given by R. That is, for every transition in $\rightarrow_P$, the number of $a/0$ constraints in the store decreases.*

Since the strict partial ordering $\succ$ is well-founded —at some point the number of $a/0$ constraints cannot further decrease— we prove that all computations must be finite and, therefore, prove program termination. □

Usually, in LP, a reduction pair on program states is defined on the basis of a reduction pair on the atoms of program states, constructing an *extension* of that reduction pair. As the way CHR states are composed is similar to LP, a same technique can be applied to the CHR-context. We introduce therefore the *multiset extension* of a reduction pair. As such, CHR states, which are multisets of constraints, can be compared on the basis of an ordering on the elements of these states.

First, we introduce the notion of a $(\succeq, \succ)$-*equivalent multisubset*.

Definition 2.3.6 ($(\succeq, \succ)$-equivalent multisubset)**.** *Let $M \uplus D$ be a multiset with universe U, on which a reduction pair $(\succeq, \succ)$ is defined. Then, D is a* $(\succeq, \succ)$-equivalent multisubset *of $M \uplus D$ iff D is non-empty and such that*

- *for every two elements a and b in D holds that $a \approx b$ and*
- *given any element a in D, no element c in M exists for which $c \approx a$.* □

Thus, the $(\succeq, \succ)$-equivalent multisubsets of a multiset represent the equivalence classes of a multiset for a reduction pair $(\succeq, \succ)$. We provide an example.

Example 2.3.4 ($(\succeq, \succ)$-equivalent multisubset)**.** *Consider a reduction pair $(\succeq, \succ)$ on a universe $U = \{a, b\}$, where $\succeq = \{(a, a), (b, b)\}$ is a pre-order relation with associated strict partial order relation $\succ = \{\}$. Then, for a multiset $M = [\![a, b, b]\!]$, $[\![a]\!]$ and $[\![b, b]\!]$ are its $(\succeq, \succ)$-equivalent multisubsets.* □

We are now ready to define the multiset extension of a reduction pair.

Definition 2.3.7 (multiset extension)**.** *Let U be a universe for the multisets M_A and M_B and let $(\succeq, \succ)$ be a reduction pair on U. Then, the* multiset

extension *of* $(\succeq, \succ)$ *is a binary relation* $\succeq_\mu$*, defined as:* $M_A \succeq_\mu M_B$ *iff for any* $(\succeq, \succ)$*-equivalent multisubset* $E = E_A \uplus E_B$ *of* $M_A \uplus M_B$*, with* $E_A \sqsubseteq M_A$ *and* $E_B \sqsubseteq M_B$*, holds that:*

> *If* $\sharp E_A < \sharp E_B$*, then a* $(\succeq, \succ)$*-equivalent multisubset* $E' = E'_A \uplus E'_B$ *of* $M_A \uplus M_B$*, with* $E'_A \sqsubseteq M_A$ *and* $E'_B \sqsubseteq M_B$*, exists, such that for any* $e \in E$ *and* $e' \in E'$ *holds that* $e' \succ e$ *and such that* $\sharp E'_A > \sharp E'_B$*.* □

The definition of a multiset extension can be confusing since it is defined in the setting of pre-orderings. Note that $>$ is the standard order over natural numbers. Thus, if for all $(\succeq, \succ)$-equivalent multisubsets $E = E_A \uplus E_B$ of $M_A \uplus M_B$, with E_A elements from M_A and E_B from M_B, the relation $\sharp E_A < \sharp E_B$ does not hold (and therefore the defined condition is trivially fulfilled), then for all $(\succeq, \succ)$-equivalent multisubsets $E = E_A \uplus E_B$ of $M_A \uplus M_B$ holds that $\sharp E_A \geq \sharp E_B$, corresponding to the intuition that $M_A \succeq_\mu M_B$.

There is a good reason for defining a multiset extension in this way. That is, not all elements in M_A or M_B must be comparable to each other, as demonstrated in the next example.

Example 2.3.5 (multiset extension)**.** *Consider a universe* $U = \{a, b, c\}$ *and a reduction pair* $(\succeq, \succ)$ *on* U*, where* $\succeq = \{(a, a), (b, b), (c, c), (a, c)\}$ *is a pre-order relation with associated strict partial order relation* $\succ = \{(a, c)\}$*. Consider two multisets* $M_A = [\![b, b, a]\!]$ *and* $M_B = [\![b, c, c, c, c, c, c]\!]$*. Since for all* $(\succeq, \succ)$*-equivalent multisubsets* $E = E_A \uplus E_B$ *of* $M_A \uplus M_B$ *holds that if* $\sharp E_A < \sharp E_B$ *—here only for* $E = E_B = [\![c, c, c, c, c, c]\!]$*— then there exists a* $(\succeq, \succ)$*-equivalent multisubset* $E' = E'_A \uplus E'_B$ *of* $M_A \uplus M_B$ *such that for* $e \in E$ *and* $e' \in E'$ *holds that* $e' \succ e$ *and* $\sharp E'_A > \sharp E'_B$ *—i.e., for* $E' = E'_A = [\![a]\!]$*—, we have that a multiset decrease* $M_A \succeq_\mu M_B$ *exists. As* $M_B \not\succeq_\mu M_A$*,* $M_A \succ_\mu M_B$*.* □

One interesting property of a multiset extension of a reduction pair is that it itself yields a reduction pair.

Proposition 2.3.1 (preservation)**.** *Let* $\succeq_\mu$ *be the multiset extension of a reduction pair* $(\succeq, \succ)$ *for a universe* U*. Then,* $(\succeq_\mu, \succ_\mu)$*, with* $\succ_\mu$ *associated to* $\succeq_\mu$*, is a reduction pair on finite multisets over* U*.* □

Proof. A multiset extension $\succeq_\mu$ is a pre-order on multisets over U. As can be verified, it is reflexive and transitive. Furthermore, the associated strict partial order $\succ_\mu$ is well-founded on *finite* multisets over U [DM79, Der82]. □

Because the multiset extension $\succeq_\mu$ of a reduction pair $(\succeq, \succ)$ on U, with associated strict partial order $\succ_\mu$, yields a reduction pair $(\succeq_\mu, \succ_\mu)$, from now

on by the multiset extension of a reduction pair $(\succeq, \succ)$ on U, we mean the reduction pair $(\succeq_\mu, \succ_\mu)$ on finite multisets over U.

A second interesting property of multiset extensions is *constructiveness*.

Proposition 2.3.2 (constructiveness)**.** *Let U be a universe for the multisets M_A, M_B and S. Let $(\succeq, \succ)$ be a reduction pair on U and $(\succeq_\mu, \succ_\mu)$ its multiset extension. Then, $M_A \succeq_\mu M_B \leftrightarrow S \uplus M_A \succeq_\mu S \uplus M_B$.* □

Proof. A proof is obtained by considering in Definition 2.3.7 that for all a, b, and $c \in \mathbb{N} : a \geq b \leftrightarrow a + c \geq b + c$. By joining both multisets M_A and M_B with S, we add to the multiplicities of M_A and M_B for any element of U the same value (see Definition 2.1.5). Therefore, the relations holding between multiplicities before adding S to both multisets hold after S has been added to both. As a result, the reason for a decrease between M_A and M_B will be the reason for a decrease between $M_A \uplus S$ and $M_B \uplus S$. The inverse is obvious. □

The importance of defining a well-founded order on program states, in terms of the atomic elements of states, now becomes clear. It allows for decrease conditions at the level of the CHR rules, or more precisely at the level of the transitions as given by the rules of the program, implying decreases between the consecutive program states in computations of the program.

Example 2.3.6 (constructiveness)**.** *Similar to Example 2.3.3, we consider a CHR program P, consisting of $(R \ @ \ a, a \Leftrightarrow a, b.)$. Then, by defining a reduction pair $(\succeq_P, \succ_P)$ on the constraints of $Atom_P$, with multiset extension $(\succeq_{\mu_P}, \succ_{\mu_P})$, such that $a \succ_P b$, we have that for R: $[\![a, a]\!] \succ_{\mu_P} [\![a, b]\!]$. Since the multiset extension is constructive, $[\![a, a]\!] \succ_{\mu_P} [\![a, b]\!] \rightarrow \forall (Q_i, Q_{i+1}) \in \rightarrow_P$: $Q_i \succ_{\mu_P} Q_{i+1}$. Thus, by well-foundedness of $\succ_{\mu_P}$, all computations of P must be finite. Therefore, P terminates for any initial CHR state.* □

2.3.3 The intended use

As programs can terminate for their intended use, but may be non-terminating when used improperly, it is interesting to develop a characterisation of the intended use and derive from it a description of the possible calls of a program to obtain more precise proof methods. We introduce for this purpose the *query set* of a CHR program.

Definition 2.3.8 (query set)**.** *Let P be a CHR program and let $\{\langle M_j, m_j \rangle \mid j \in J\}$ be a set of initial CHR states of interest for P. Then, the set $I = \bigcup_{j \in J} M_j$ is called a* query set *for P.* □

A query set characterises the intended use of a CHR program only to some extent, e.g., there is no characterisation of the relations that must hold between constraints present in initial CHR states. However, on the basis of a query set I, we can over-approximate all possible initial CHR states as $\mathcal{I} = \{\langle I, \mu_Q\rangle \mid \mu_Q : I \to \mathbb{N}\}$, where we use I as a universe for these multisets.

Considering a query set leads to a characterisation of the possible calls that can occur during execution of a CHR program.

Definition 2.3.9 (call set)**.** *Let P be a CHR program and I a query set for P. For any initial CHR state $Q \in \mathcal{I}$, the* call set *$Call(P, Q)$ w.r.t. Q is the closure under substitution of the set containing all built-in and CHR constraints C for which a variant of C is either used to call CT (including guards) or* used *to fire a CHR rule of P, in some computation of P for Q. Then, the call set $Call(P, I)$ w.r.t. I is the set $\bigcup_{Q\in\mathcal{I}} Call(P, Q)$.* □

The concepts of a query and call set originate from LP-termination and yield safe over-approximations of the actual queries and calls occurring in computations. In CHR there can be two different notions for the call set. We can consider the constraints *added* in computations or the constraints *used* in computations. The latter notion, as used above in Definition 2.3.9, is more precise, since it allows consideration of matching and guard satisfaction.

In single-headed LP, the notions of query and call set yield more accurate descriptions when compared to CHR [Ngu09]. The atoms in the call set correspond directly to actual calls to the program while, in CHR, this is, in the presence of multi-headed rules, only partially the case. For termination, however, as shown in [PDS09a], this representation is sufficiently accurate.

Finally, notice that the call set includes guard and added built-in constraints and that under assumption of a terminating host language, that cannot introduce CHR constraints, we could have ignored them instead. However, in regard to automating termination proofs for CHR, where typically termination is also proven of the host language (see Chapter 6), we prefer to include them to improve precision of the termination analysis of the host language.

Example 2.3.7 (intended use)**.** *Most often, query sets and call sets are infinite. Considering Example 2.2.1 —with $[\![msort([s(s(s(0))), 0, s(s(0)), s(0)])]\!]$ as a possible initial CHR state— the intended initial CHR states for the merge-sort program consist of a single $msort/1$ atom, with at its argument position a list of 2^n ground terms of the form $s(s(\ldots s(0)))$.*

The query set I is the infinite set of all atoms of this form. The possible initial CHR states $\mathcal{I}$ are all multisets definable with elements from I. These are more general than the intended initial states —e.g., an initial CHR state

with multiple occurrences of an msort/1 constraint is defined— but remain sufficiently precise for proving termination (see Section 2.4.2).

The call set Call(P, I) is the set $\{msort(l),\ r(t_1,t_2),\ a(t_3,t_4),\ less(t_5,t_6)\ \mid$ *l is a bounded list of terms* t_0 *and* $t_0,t_1,\ldots,t_6$ *are terms of the form* $s(s(\ldots s(0)))\}$. *Contrary to the query set, the call set contains msort/1 constraints with lists of any length. This is due to the second rule of the Mergesort program. It is applicable on msort/1 constraints with a list of any length, recursively decomposing the list one element at the time.* □

Knowledge of possible calls of a program also provides information on the instantiated parts of calls, information which is essential for proving termination of non-ground programs. We elaborate on this second motivation for introducing call sets in the next subsection.

2.3.4 Termination of non-ground CHR programs

To prove termination of non-ground programs, we need to be sure to only consider the parts of constraints that are instantiated in calls to the program. In CHR, although introduced through the host language, there is backpropagation of data and thus if constraints in a constraint store contain uninstantiated parts, they may get instantiated as a consequence of bindings which are propagated backward. Since this may change the order of a constraint, to guarantee termination using local conditions on rules, we need to consider *rigidity* of the constraints used in calls w.r.t. the chosen ordering on constraints [DSS02].

Definition 2.3.10 (rigidity)**.** *A (term or) constraint C is called* rigid *w.r.t. a reduction pair* $(\succeq,\succ)$ *iff* $C \approx C\sigma$ *holds for any substitution* σ. *A set or multiset of (terms or) constraints S is called rigid w.r.t.* $(\succeq,\succ)$ *iff all elements of S are rigid w.r.t.* $(\succeq,\succ)$. □

In the next example, we demonstrate rigidity for the orderings given by *term-size* and *list-length* [BCF91]. Therefore, we first introduce these concepts.

Definition 2.3.11 (list-length)**.** *Consider a term t, then the* list-length *of t, denoted* $|t|_{ll}$, *is* $1+|tr|_{ll}$ *if* $t=[te|tr]$ *and* 0 *otherwise.* □

For instance, $[s(0),0] \succ [s(s(s(0)))]$ if $\succeq$ is given by list-length.

Definition 2.3.12 (term-size)**.** *Consider a term t, then the* term-size *of t, denoted* $|t|_{ts}$, *is* $1+\sum_{i=1}^{n}|t_i|_{ts}$ *if* $t=f(t_1,\ldots,t_n)$ $(n\geq 1)$ *and* 0 *otherwise.* □

For instance, $[s(s(s(0)))] \succ' [s(0),0]$ if $\succeq'$ is given by term-size.

Example 2.3.8 (rigidity)**.** *The list* $[X|t]$*, with* X *a variable and* t *a ground term, is rigid w.r.t. the reduction pair* $(\succeq, \succ)$ *corresponding to list-length. For further instantiations of* $[X|t]$*, given by the substitution* σ*, we have that* $[X|t] \approx [X|t]\sigma$ *as only the first element of the list can get further instantiated and thus the length of the list cannot alter. The list* $[X|t]$ *is however not rigid w.r.t. the reduction pair* $(\succeq', \succ')$ *corresponding to term-size. Term-size measures the entire size of the term which represents the list instead of only the length of the list, and because further instantiations of* $[X|t]$ *can result in an increase in term-size,* $[X|t]$ *is not necessarily order equivalent to* $[X|t]\sigma$*.* □

Note that we prefer conditions on the basis of rigidity and not *boundedness* like in [Frü00]. In practice, rigidity is easier to verify [BCF91]. Note however that we will not require rigidity of $Atom_P$ as this would result in reduction pairs for which the constraints of some predicate are all order equivalent. Instead, we construct reduction pairs on $Call(P, I)$ for which $Call(P, I)$ is rigid (see Section 2.3.6) and extend these reduction pairs to reduction pairs on $Atom_P$ (see Definition 2.3.17) for which $Atom_P$ is bounded (see Lemma 2.3.2).

2.3.5 The success set of a CHR program

When formulating termination conditions for the CHR rules of a program, we can increase the accuracy of our termination proofs by considering satisfaction of the guards of these rules. Since guards can introduce bindings, their effect can be considered in decrease conditions, resulting in increased precision.

Example 2.3.9 (intermediate calls 1)**.** *Consider a similar rule to the CHR rule introduced in Example 1.3.1:*

$$gcd(M), gcd(N) \Leftrightarrow N >= M, M > 0, L \text{ is } N - M \mid gcd(M), gcd(L).$$

It only differs from the second CHR rule of Example 1.3.1 in that the built-in constraint L *is* $N - M$ *is moved from the body of the rule to the guard of the rule such that it is now solved deterministically. The rule activates on ground* $gcd/1$ *constraints containing positive integers. By considering the effect of the call* L *is* $N - M$ *—i.e.,* M *subtracted from* N *yields* L*— together with the guarantee that* $N >= M$ *and* $M > 0$*, we can prove that a strict decrease exists between the value of* N *and the value of* L *using the standard ordering on natural numbers. Therefore, a strict multiset decrease can be shown to exist between the removed head and the body of the rule for any transition using the rule and thus we can prove program termination.* □

The effect of an *intermediate call* can be estimated using the *success set* of its predicate. In LP, the success set is the procedural counterpart of the least Herbrand model [Llo87].

Definition 2.3.13 (success set of an LP program)**.** *Let P be an LP program. The success set R_P^{SS} of P is the set of all $A \in B_P$ such that $P \cup \{\leftarrow A\}$ has a refutation. The success set $R_{p/n}^{SS}$ of a predicate p/n of P is the set $\{A \mid A \in R_P^{SS} \wedge rel(A) = p/n\}$.* □

In practice, the success set of a predicate is often given as a safe over-approximation of its actual contents. In the next example, we discuss the success set of the built-in predicates of the CHR program of Example 2.3.9.

Example 2.3.10 (intermediate calls 2)**.** *Revisiting Example 2.3.9, then the success set $R_{>=/2}^{SS}$ of $>=/2$ is $\{N >= M \mid N, M \in \mathbb{N} \wedge N \geq M\}$. For the other built-in constraints, we have $R_{>/2}^{SS} = \{N > M \mid N, M \in \mathbb{N} \wedge N > M\}$ and $R_{is/2}^{SS} = \{L \text{ is } N - M \mid N, M, L \in \mathbb{N} \wedge L = N - M\}$, where $is/2$ is the restricted version of the Prolog predicate $is/2$, adapted to its use in P. When considering the CHR rule of Example 2.3.9, we can now restrict our attention to those instantiations of the rule for which the guards are part of their success set: $\forall N, M, L \in \mathbb{N} : N >= M \in R_{>=/2}^{SS} \wedge M > 0 \in R_{>/2}^{SS} \wedge L \text{ is } N - M \in R_{is/2}^{SS}$. For these instances, we can guarantee a strict multiset decrease between the head and body of the rule, as argued in Example 2.3.9.* □

Next to the built-in constraints of the guard of a CHR rule, it may also be the case that added built-in constraints, in the body of a CHR rule, need to be considered to prove decreases between the head and the body of a CHR rule.

Example 2.3.11 (intermediate calls 3)**.** *Consider the CHR rule*

$$gcd(M), gcd(N) \Leftrightarrow N >= M, M > 0 \mid L \text{ is } N - M, gcd(M), gcd(L).$$

of Example 1.3.1. As the built-in constraint L is $N - M$ is part of the body of the CHR rule, in contrast to Example 2.3.9, it is solved non-deterministically. Given termination of the host language, when L is $N - M$ is solved, it returns bindings similar as in Example 2.3.9. Thus, we can prove termination if it is guaranteed that this built-in constraint is solved before *the added CHR constraint $gcd(L)$ can be used to fire the CHR rule.*

Because of the guard of the CHR rule, only $gcd(L)$ constraints with L a natural number can fire the rule. Since bindings for L are generated as a direct consequence of solving the added L is $N - M$ constraints, we can consider L is $N-M$ as an intermediate call for $gcd(L)$. Therefore, to prove termination, we may estimate the possible values for $is/2$ constraints by the success set of their predicate like in Example 2.3.9. □

Estimating the effect of added built-in constraints for proving termination is in general not correct. Even though we assume a terminating host language, a built-in constraint can be delayed forever and thus might never be evaluated. Therefore, a built-in constraint can be added that would fail the program if solved, without actually resulting in program failure. To illustrate this, consider the CHR rule $(a \Leftrightarrow 4 \; is \; 3+2, a.)$. For an initial CHR state $[\![a]\!]$, the rule can repeatedly replace the constraint a without solving any of the added built-in constraints. In this case, the program runs forever.

To overcome this problem, we require *fairness*. That is, no constraint can be delayed forever. As it is reasonable to consider a fair semantics [Frü98], from now on, any CHR semantics of interest is considered to be *fair*. Therefore, by assumption of a terminating host language, a built-in constraint will either finitely fail the program or be bound according to its (*extended*) success set (see Definition 2.3.15) in a finite number of steps.

Note that if we would have considered the call set to be the constraints *added* by CHR rules, in contrast to the constraints *used* as in Definition 2.3.9, there would be no increment in considering the effect of the body built-in constraints due to a requirement of rigidity of the call set (see Section 2.3.4).

For a CHR program P (and its host language), we define the success set as the set of ground constraints that turn up in successful CHR computations of P.

Definition 2.3.14 (success set of a CHR program)**.** *Let P be a CHR program. The success set R_P^{SS} of P is the set of all $A \in B_P$ such that there exists a CHR state $[\![A]\!] \uplus S$, with S possibly empty, that is an initial CHR state in a successful computation of P. The success set $R_{c/n}^{SS}$ of a predicate c/n is the set $\{A \mid A \in R_P^{SS} \wedge rel(A) = c/n\}$.* □

Example 2.3.12 (success set of a CHR program)**.** *Consider a CHR program*

$$diff(A, B, C) \Leftrightarrow C \; is \; A - B.$$

Similar to Example 2.3.10, using Definition 2.3.14 instead, we can derive for $is/2$ the same success set as in Example 2.3.10. As the $diff/3$ constraints can all fire the CHR rule, adding an $is/2$ built-in constraint with the same arguments, we can derive for $diff/3$ that $R_{diff/3}^{SS} = \{diff(N, M, L) \mid N, M, L \in \mathbb{N} \wedge L = N - M\}$.

Consider now a CHR program P:

$$a(A), b(B), c(C) \Leftrightarrow C \; is \; A - B.$$

Here, presence of a constraint matching a head constraint of the above CHR rule does not guarantee rule application. There is also a requirement for partner

constraints. Therefore, all ground instances of $a(A)$, $b(B)$ *and* $c(C)$, *in* B_P, *are in their success set.* □

There is a good reason for introducing the success set of a CHR program and not restricting to the success set of the host language of the CHR program. That is, we do not wish to be concerned with host language specifics and thus do not wish to define termination conditions for every host language of interest. Since the success set of the host language is contained within the success set of a CHR program, this resolves the problem. For a CHR program with host language LP, we have the following property.

Proposition 2.3.3 (containment)**.** *Let* P *be a CHR program with a constraint theory* CT *defined by an LP program* P'. *Then, for any built-in constraint* b: $b \in R^{SS}_{P'} \leftrightarrow b \in R^{SS}_{P}$. □

Proof. Assume a built-in constraint $b \in R^{SS}_{P'}$. Then, b has a refutation in P'. Thus, solving $[\![b]\!]$ by P' results in a successful computation of P and thus $b \in R^{SS}_{P}$. Assume a built-in constraint $b \in R^{SS}_{P}$. For any initial CHR state $[\![b]\!] \uplus S$ in a successful computation of P, b must be successfully solved by P'. Therefore, b has a refutation in P' and thus $b \in R^{SS}_{P'}$. □

A second motivation for introducing the success set of a CHR program and not only for the host language could be that also CHR constraints can behave as intermediate calls, as we demonstrate in the next example.

Example 2.3.13 (intermediate calls 4)**.** *Consider a CHR program* P:

$$gcd(M), gcd(N) \Leftrightarrow N >= M, M > 0 \mid diff(N, M, L), gcd(M), gcd(L).$$
$$diff(A, B, C) \Leftrightarrow C \; is \; A - B.$$

Here, in order to prove termination of the program, we need to estimate the effect of the CHR constraint $diff(N, M, L)$ *in the first rule, because, similar to Example 2.3.11, it is an intermediate call for* $gcd(L)$. *Given the second rule, we know that the effect of* $diff(N, M, L)$ *coincides with that of* $L \; is \; N - M$ *(see Example 2.3.12). Together with the effect of the guards of the first rule, we thus know that, in the first rule,* L *must have a strictly smaller value than* N *and, therefore, that we can show a strict multiset decrease between the head and the body of the CHR rule.* □

As discussed in Section 5.2.1, using the success set of a CHR predicate is not straightforward. Secondly, CHR programs where the success set of CHR predicates is required to prove termination do not often occur in practice as the availability of a host language disfavours it. Therefore, we will not consider CHR constraints as candidates for intermediate calls.

2.3.6 The Ranking Condition for abstract CHR

The *Ranking Condition* (RC) for an abstract CHR program is similar to the decrease conditions for an LP program [DSD94]. Abstract CHR rules are in fact multi-headed variants of LP-clauses. For them to induce a decrease at the state-size level, there must exist a decrease between the multiset of constraints removed from the constraint store and the multiset of constraints added.

First, note that in CHR computations not all built-in constraints, when solved successfully, need to produce bindings for each of the variables in these built-in constraints. The next example illustrates this.

Example 2.3.14 (CHR success set)**.** *Consider a built-in predicate $less/2$, with the following Prolog implementation:*

$$less(s(X), s(Y)) :- less(X, Y). \qquad less(0, s(Y)).$$

For $less/2$, a call $less(s(s(0)), s(s(s(A))))$, with A a variable, succeeds without generating bindings for A. Clearly, $less(s(s(0)), s(s(s(A))))$ is not part of the success set $R^{SS}_{less/2}$ of $less/2$, however, any ground instance thereof is. □

During CHR computations, therefore, constraints that are successfully solved belong to the *extended success set* of a CHR program.

Definition 2.3.15 (extended success set)**.** *Let P be a CHR program and R^{SS}_P the success set of P. The* extended success set *of P is the set $R^{SSE}_P = \{A \mid A \in Atom_P \wedge$ for every grounding substitution θ of $A : A\theta \in R^{SS}_P\}$. The extended success set $R^{SSE}_{c/n}$ of a predicate c/n is $\{A \mid A \in R^{SSE}_P \wedge rel(A) = c/n\}$.* □

For the same reason as to why we only discussed the abstract CHR semantics in terms of simplification (see Section 2.2), we only define the RC for abstract CHR in terms of simplification and not propagation or simpagation.

Definition 2.3.16 (RC for abstract CHR)**.** *Let P be a CHR program and I a query set. Let $(\succeq, \succ)$ be a reduction pair on $Call(P, I)$ such that $Call(P, I)$ is rigid w.r.t. $(\succeq, \succ)$ and let $(\succeq_\mu, \succ_\mu)$ be the multiset extension of $(\succeq, \succ)$. Then, a CHR program P for I satisfies the* RC for abstract CHR *iff for any substitution χ of the variables of a simplification rule*

$$R \ @ \ H_1, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

in P, such that

$$\forall H \in [\![H_1, \ldots, H_n]\!]\chi : H \in Call(P, I),$$

$$\forall G \in [\![G_1, \ldots, G_k]\!]\chi : G \in R^{SSE}_{rel(G)}, \text{ and}$$

$$\forall B \in [\![B_1, \ldots, B_l]\!]\chi : B \in R^{SSE}_{rel(B)},$$

the decrease

$$[\![H_1, \ldots, H_n]\!]\chi \succ_\mu [\![C \mid C \in [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi \wedge C \in Call(P, I)]\!]$$

holds. □

Note that we defined the RC in terms of an ordering on the elements of the call set $Call(P, I)$ and not in terms of an ordering on the elements of $Atom_P$. The reason for this is that any element in $Atom_P$, outside $Call(P, I)$, can always be ordered strictly smaller than the elements in $Call(P, I)$. For this purpose, we introduce the *natural extension* of a reduction pair.

Definition 2.3.17 (natural extension)**.** *Let S_1 and S_2 be two sets of constraints such that $S_1 \subseteq S_2$. Let $\succeq$ be a pre-order relation on S_1. Then, the* natural extension *of $\succeq$ to S_2 is $\succeq'$ on S_2, defined as $\succeq' = \succeq \cup \succeq_\delta$, where $\succeq_\delta = \{(a, b), (b, b) \mid a \in S_1,\ b \in S_2 \setminus S_1\} \cup \{(b, b'), (b', b) \mid b, b' \in S_2 \setminus S_1\}$.* □

By construction, the natural extension of a reduction pair is a reduction pair.

Proposition 2.3.4 (preservation)**.** *Let S_1 and S_2 be two sets of constraints such that $S_1 \subseteq S_2$. Let $\succeq$ be a pre-order relation on S_1 and $\succeq'$ its natural extension to S_2. Then, $(\succeq, \succ)$ is a reduction pair iff $(\succeq', \succ')$ is a reduction pair, with $\succ$ and $\succ'$ the strict partial orderings associated to $\succeq$ and $\succeq'$.* □

Proof. As can be verified, $\succeq'$ is a pre-order iff $\succeq$ is a pre-order. The associated partial order $\succ'$ is well-founded if $\succ$ is well-founded as there exists no (a, b) in $\succ'$ such that a and b are both in S_2 and not in S_1. Obviously, if $\succ'$ is well-founded, then $\succ$ is well-founded. □

Therefore, we may also say that a reduction pair $(\succeq', \succ')$ is the natural extension of a reduction pair $(\succeq, \succ)$. Thus, considering the natural extension of a reduction pair $(\succeq, \succ)$ on $Call(P, I)$ to a reduction pair $(\succeq_P, \succ_P)$ on $Atom_P$, we preserve the order relations for elements in the call set, while elements outside the call set are all order equivalent and ordered strictly smaller than any of the constraints in the call set.

Using a natural extension, we can reformulate the RC for abstract CHR.

Corollary 2.3.1 (RC for abstract CHR)**.** *Let P be a CHR program and I a query set. Furthermore, let $(\succeq, \succ)$ be a reduction pair on $Call(P, I)$ such that $Call(P, I)$ is rigid w.r.t. $(\succeq, \succ)$, let $(\succeq_P, \succ_P)$ be the natural extension of $(\succeq, \succ)$ to $Atom_P$ and let $(\succeq_{\mu_P}, \succ_{\mu_P})$ be the multiset extension of $(\succeq_P, \succ_P)$. Then, a CHR program P for I satisfies the* RC for abstract CHR *w.r.t. $(\succeq, \succ)$ iff for any substitution χ of the variables of a simplification rule*

$$R \ @ \ H_1, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

in P, such that

$$\forall H \in [\![H_1, \ldots, H_n]\!]\chi : H \in Call(P, I),$$

$$\forall G \in [\![G_1, \ldots, G_k]\!]\chi : G \in R^{SSE}_{rel(G)}, \text{ and}$$

$$\forall B \in [\![B_1, \ldots, B_l]\!]\chi : B \in R^{SSE}_{rel(B)},$$

the decrease $[\![H_1, \ldots, H_n]\!]\chi \succ_{\mu_P} [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi$ holds. □

Proof. Consider a reduction pair $(\succeq, \succ)$ on $Call(P, I)$ and its natural extension $(\succeq_P, \succ_P)$ to $Atom_P$ (see Definition 2.3.17). Furthermore, let $(\succeq_\mu, \succ_\mu)$ and $(\succeq_{\mu_P}, \succ_{\mu_P})$ be the multiset extensions of $(\succeq, \succ)$ and $(\succeq_P, \succ_P)$, respectively. Then, by natural extensions and multiset extensions, because the head constraints $[\![H_1, \ldots, H_n]\!]\chi$, which are part of the call set, must be strictly greater $(\succ_P)$ than the body constraints not in the call set, we have that

$$[\![H_1, \ldots, H_n]\!]\chi \succ_\mu [\![C \mid C \in [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi \wedge C \in Call(P, I)]\!]$$

holds iff

$$[\![H_1, \ldots, H_n]\!]\chi \succ_{\mu_P} [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi$$

holds. □

Notice that $Atom_P$ is not rigid w.r.t. $(\succeq_P, \succ_P)$ if $Call(P, I)$ is rigid w.r.t. $(\succeq, \succ)$, i.e., the natural extension does not preserve rigidity. Consider for example some constraint $a(A)$ not in the call set and a more instantiated constraint $a(s(0))$ in the call set. Then, by construction, $a(s(0)) \succ_P a(A)$.

Finally, note that built-in constraints can be ignored in the decrease conditions. This is because built-in constraints (by assumption) cannot introduce new CHR constraints. Therefore, built-in constraints of the call set can always be ordered strictly smaller than CHR constraints of the call set. Also note that for simpagation rules we can ignore the kept heads, since kept heads will appear on both sides of the decrease conditions (see Proposition 2.3.2).

2.3.7 Correctness of the RC for abstract CHR

To prove the correctness of the RC for abstract CHR (see Definition 2.3.16), i.e., satisfaction of the RC implies program termination, we first introduce a representation of CHR states, defined in terms of CHR computations.

Definition 2.3.18 (abstract CHR state representation)**.** *Let P be an abstract CHR program and I a query set. Let $Q_0 \xrightarrow{P,\phi_1} Q_1 \xrightarrow{P,\phi_2} Q_2 \xrightarrow{P,\phi_3} \dots \xrightarrow{P,\phi_q} Q_q \xrightarrow{P,\phi_{q+1}} \dots$ be any computation of P for I, where $\phi_i = \theta_i$ —the answer substitution for the built-in constraint— if $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ is a solve transition, and $\phi_i = \sigma_i\theta_i$ —the composition of the match substitution and the answer substitution for the guard— if $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ is a simplification transition. Then, for $q \geq i$, we define*

- $\phi_i^q = \phi^\emptyset\phi_{i+1}\phi_{i+2}\dots\phi_q$, *with $\phi^\emptyset$ the empty substitution, and*
- $|Q_i|^q = Q_i\phi_i^q$. □

We will also need the following lemma.

Lemma 2.3.2 (boundedness of $|.|^q$ in q)**.** *Let P be an abstract CHR program and I a query set. Let $Q_0 \xrightarrow{P,\phi_1} Q_1 \xrightarrow{P,\phi_2} Q_2 \xrightarrow{P,\phi_3} \dots \xrightarrow{P,\phi_q} Q_q \xrightarrow{P,\phi_{q+1}} \dots$ be any computation of P for I. Let $(\succeq, \succ)$ be a reduction pair on $Call(P,I)$, for which $Call(P,I)$ is rigid, and let $(\succeq_P, \succ_P)$ be its natural extension to $Atom_P$. Furthermore, let $(\succeq_{\mu_P}, \succ_{\mu_P})$ be the multiset extension of $(\succeq_P, \succ_P)$. Then, for any Q_i in the computation, there exists a $q \geq i$, with Q_q in the computation, such that for all $q' \geq q$, with $Q_{q'}$ in the computation, holds that $|Q_i|^q \approx_{\mu_P} |Q_i|^{q'}$.* □

Proof. Q_i is a finite multiset of constraints in $Atom_P$. Let A be a constraint in Q_i. Furthermore, consider that

- by rigidity, if $A \in Call(P,I)$, then for any $q \geq i$: $A\phi_i^q \approx_P A$, and
- by natural extensions, if $A \in Atom_P \setminus Call(P,I)$, then
  for any $q \geq i$ such that $A\phi_i^q \in Atom_P \setminus Call(P,I) : A\phi_i^q \approx_P A$, and
  for any $q \geq i$ such that $A\phi_i^q \in Call(P,I) : A\phi_i^q \succ_P A$.

So, for any $A \in Q_i$, either for all $q \geq i$: $A\phi_i^q \approx_P A$, or there exists a $q_A \geq i$ such that $A\phi_i^{q_A} \succ_P A$ and for all $q' \geq q_A$: $A\phi_i^{q'} \approx_P A\phi_i^{q_A}$. Since Q_i is finite, by taking the maximum of all such q_A, we obtain the result. □

Using the abstract CHR state representation, under satisfaction of the RC for abstract CHR, we can prove that for any transition in a non-failed computation, there exists a strict decrease between the consecutive program states. We have the following lemma.

Lemma 2.3.3. *Let P be an abstract CHR program which satisfies the RC for abstract CHR for a query set I w.r.t. $(\succeq, \succ)$ on $Call(P, I)$. Let $(\succeq_P, \succ_P)$ be the natural extension of $(\succeq, \succ)$ to $Atom_P$ and $(\succeq_{\mu_P}, \succ_{\mu_P})$ the multiset extension of $(\succeq_P, \succ_P)$. Then, for any transition $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ in a non-failed computation $Q_0 \xrightarrow{P,\phi_1} Q_1 \xrightarrow{P,\phi_2} Q_2 \xrightarrow{P,\phi_3} \ldots \xrightarrow{P,\phi_q} Q_q \xrightarrow{P,\phi_{q+1}} \ldots$ of P for I, holds that $\exists q \geq i$ such that $\forall q' \geq q : |Q_{i-1}|^{q'} \approx_{\mu_P} |Q_{i-1}|^q \succ_{\mu_P} |Q_i|^q \approx_{\mu_P} |Q_i|^{q'}$.* □

Proof. There are two kinds of transitions to consider:

1. Let $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ be a solve transition in a non-failed computation of P for I and let Q_q be in the computation, with $q \geq i$. Then, necessarily,

   - $|Q_{i-1}|^q = Q_{i-1}\phi_{i-1}^q = (\llbracket b \rrbracket \uplus S)\phi_{i-1}^q$,
   - $|Q_i|^q = Q_i\phi_i^q = (S\theta_i)\phi_i^q$, and
   - $\phi_{i-1}^q = \theta_i\phi_i^q$.

   As $\llbracket b \rrbracket\phi_{i-1}^q \succ_{\mu_P} [\![]\!]$, by constructiveness, $\llbracket b \rrbracket\phi_{i-1}^q \uplus S\phi_{i-1}^q \succ_{\mu_P} S\phi_{i-1}^q$, and thus $|Q_{i-1}|^q \succ_{\mu_P} |Q_i|^q$.

   Since this holds for any $q \geq i$, and by Lemma 2.3.2 there exists a q_{i-1}, such that for all $q' \geq q_{i-1}$, $|Q_{i-1}|^{q'} \approx_{\mu_P} |Q_{i-1}|^{q_{i-1}}$, and a q_i, such that for all $q' \geq q_i$, $|Q_i|^{q'} \approx_{\mu_P} |Q_i|^{q_i}$, we have that for $q = max\{q_{i-1}, q_i\}$, for all $q' \geq q : |Q_{i-1}|^{q'} \approx_{\mu_P} |Q_{i-1}|^q \succ_{\mu_P} |Q_i|^q \approx_{\mu_P} |Q_i|^{q'}$.

2. Let $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ be a simplification transition in a non-failed computation of P for I. Let $(R \ @ \ H_1, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.)$ be the simplification rule of P applied in the transition and this on the CHR constraints $h_1, \ldots, h_n$ of Q_{i-1} with match substitution σ_i, such that the guards $G_1, \ldots, G_k$ all succeed with answer substitution θ_i. Let furthermore Q_q be in the computation, with $q \geq i$. Then, necessarily,

   - $|Q_{i-1}|^q = Q_{i-1}\phi_{i-1}^q = (\llbracket h_1, \ldots, h_n \rrbracket \uplus S)\phi_{i-1}^q$,
   - $|Q_i|^q = Q_i\phi_i^q = ((\llbracket B_1, \ldots, B_l, C_1, \ldots, C_m \rrbracket \uplus S)\sigma_i\theta_i)\phi_i^q$, and
   - $\phi_{i-1}^q = \sigma_i\theta_i\phi_i^q$.

Let $\chi^q_{i-1} = \sigma_i\theta_i\chi^q_i$ be the substitution obtained by restricting the domain of $\phi^q_{i-1} = \sigma_i\theta_i\phi^q_i$ to the variables of R. Then, considering the precondition of the RC for abstract CHR (see Corollary 2.3.1), since both the call set and the extended success set are closed under substitution, we have that for any $q \geq i$:

$$\forall H \in [\![H_1, \ldots, H_n]\!]\chi^q_{i-1} : H \in Call(P, I), \text{ and}$$
$$\forall G \in [\![G_1, \ldots, G_k]\!]\chi^q_{i-1} : G \in R^{SSE}_{rel(G)}.$$

As furthermore, by fairness, there must exists a state Q_{q_f} in the considered computation, with $q_f \geq i$, such that the body built-in constraints, $B_1, \ldots, B_l$, of R were all solved successfully, we have that for any $q' \geq q_f$:

$$\forall B \in [\![B_1, \ldots, B_l]\!]\chi^{q'}_{i-1} : B \in R^{SSE}_{rel(B)}.$$

Therefore, a $q_f \geq i$ exists, such that for any $q' \geq q_f$, $\chi^{q'}_{i-1}$ satisfies the precondition of the RC for abstract CHR. Therefore, under satisfaction of the RC for abstract CHR, this implies that a $q_f \geq i$ exists, such that for any $q' \geq q_f : [\![H_1, \ldots, H_n]\!]\chi^{q'}_{i-1} \succ_{\mu_P} [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi^{q'}_{i-1}$, or else, $[\![H_1, \ldots, H_n]\!]\phi^{q'}_{i-1} \succ_{\mu_P} [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\phi^{q'}_{i-1}$. By constructiveness, $[\![H_1, \ldots, H_n]\!]\phi^{q'}_{i-1} \uplus S\phi^{q'}_{i-1} \succ_{\mu_P} [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\phi^{q'}_{i-1} \uplus S\phi^{q'}_{i-1}$, and thus

$$[\![h_1, \ldots, h_n]\!]\phi^{q'}_{i-1} \uplus S\phi^{q'}_{i-1} \succ_{\mu_P} [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\phi^{q'}_{i-1} \uplus S\phi^{q'}_{i-1}.$$

Thus, there exists a $q_f \geq i$, such that for any $q' \geq q_f : |Q_{i-1}|^{q'} \succ_{\mu_P} |Q_i|^{q'}$. By the same argument as used at the end of part 1, there also exists a q, such that for all $q' \geq q : |Q_{i-1}|^{q'} \approx_{\mu_P} |Q_{i-1}|^q \succ_{\mu_P} |Q_i|^q \approx_{\mu_P} |Q_i|^{q'}$.

□

We are ready to express the correctness of our RC, i.e., the RC for abstract CHR guarantees termination of the CHR program P for the query set I.

Theorem 2.3.1. *If an abstract CHR program P for a query set I satisfies the RC for abstract CHR, then P is terminating for I.* □

Proof. We prove that all computations of P for I are finite by proving that all non-failed computations of P for I are finite. Then, by Lemma 2.3.1, since all computations are finite, P is terminating for I.

If P for I satisfies the RC for abstract CHR, then, by Lemma 2.3.3, for any non-failed computation $Q_0 \xrightarrow{P,\phi_1} Q_1 \xrightarrow{P,\phi_2} Q_2 \xrightarrow{P,\phi_3} \ldots \xrightarrow{P,\phi_n} Q_n \xrightarrow{P,\phi_{n+1}} \ldots$ of P for I, there exist $q^0 \leq q^1 \leq q^2 \leq \cdots \leq q^n \leq \ldots$ such that

$$|Q_0|^{q^0} \succ_{\mu_P} |Q_1|^{q^0} \approx_{\mu_P} |Q_1|^{q^1} \succ_{\mu_P} |Q_2|^{q^1} \approx_{\mu_P}$$

$$|Q_2|^{q^2} \succ_{\mu_P} |Q_3|^{q^2} \approx_{\mu_P} \cdots \approx_{\mu_P} |Q_n|^{q^n} \succ_{\mu_P} |Q_{n+1}|^{q^n} \approx_{\mu_P} \ldots.$$

Let Σ_P be the set of abstract CHR states of P. Let $(\succeq'_{\mu_P}, \succ'_{\mu_P})$ be the reduction pair, associated to $(\succeq_{\mu_P}, \succ_{\mu_P})$, on the set $\Sigma_P/\approx_{\mu_P}$, i.e., the equivalence classes of Σ_P w.r.t. $(\succeq_{\mu_P}, \succ_{\mu_P})$. For any $s \in \Sigma_P$, we denote by $[s]$ the corresponding element of $\Sigma_P/\approx_{\mu_P}$. Then, for any non-failed computation of P for I, we can construct a sequence,

$$[|Q_0|^{q^0}] \succ'_{\mu_P} [|Q_1|^{q^0}] \succ'_{\mu_P} [|Q_2|^{q^1}] \succ'_{\mu_P} \cdots \succ'_{\mu_P} [|Q_{n+1}|^{q^n}] \succ'_{\mu_P} \ldots,$$

of elements of $\Sigma_P/\approx_{\mu_P}$.

If the computation is infinite, we obtain an infinite strict decreasing chain of elements of $\Sigma_P/\approx_{\mu_P}$. As this contradicts the well-foundedness of $\succ'_{\mu_P}$ on $\Sigma_P/\approx_{\mu_P}$, any non-failed computation of P for I must be finite. □

2.4 Termination of typical abstract CHR programs

In this section, we discuss termination of typical abstract CHR programs by application of the RC for abstract CHR (see Definition 2.3.16).

2.4.1 Greatest common divisor

Recall the CHR program P from Example 1.3.1 for computing the greatest common divisor:

R_1 @ $gcd(0) \Leftrightarrow true$.
R_2 @ $gcd(M), gcd(N) \Leftrightarrow N >= M, M > 0 \mid L \; is \; N - M, gcd(M), gcd(L)$.

For any number of $gcd/1$ constraints with positive integer arguments, the program returns an answer state containing a single $gcd/1$ constraint. Given a query set $I = \{gcd(t) \mid t \in \mathbb{N}\}$, we can derive a call set $Call(P, I) = I \cup \{t_1 >= t_2, t_3 > 0, t_6 \; is \; t_4 - t_5 \mid t_1, t_2, \ldots, t_5 \in \mathbb{N} \wedge t_6 \in Var_P\}$.

To prove termination of P for I, we can order the $gcd/1$ constraints of the call set by their argument values using the reduction pair $(\geq, >)$ on positive integer

numbers. E.g., $gcd(5) \succ gcd(3)$. Furthermore, all built-in constraints of the call set are order equivalent and strictly smaller than any of the $gcd/1$ constraints of the call set, such that they can be ignored in the decrease conditions of the RC for abstract CHR. It can be verified that $(\succeq, \succ)$ is a reduction pair and that $Call(P, I)$ is rigid w.r.t. $(\succeq, \succ)$.

Recall the success sets of the built-in constraints of P from Example 2.3.10:

$$R^{SS}_{>=/2} = \{N >= M \mid N, M \in \mathbb{N} \wedge N \geq M\},$$
$$R^{SS}_{>/2} = \{N > M \mid N, M \in \mathbb{N} \wedge N > M\} \text{ and}$$
$$R^{SS}_{is/2} = \{L \; is \; N - M \mid N, M, L \in \mathbb{N} \wedge L = N - M\}.$$

It can be verified that $R^{SSE}_{>=/2} = R^{SS}_{>=/2}$, $R^{SSE}_{>/2} = R^{SS}_{>/2}$ and $R^{SSE}_{is/2} = R^{SS}_{is/2}$.

Let $(\succeq_\mu, \succ_\mu)$ be the multiset extension of $(\succeq, \succ)$. Then, P for I satisfies the RC for abstract CHR if for any substitution χ of the variables of R_1 holds that

$$gcd(0)\chi \in Call(P, I) \rightarrow [\![gcd(0)]\!]\chi \succ_\mu [\![C \mid C \in [\![]\!]\chi \wedge C \in Call(P, I)]\!] \; and$$

for any substitution χ of the variables of R_2 holds that

$$gcd(M)\chi \in Call(P, I) \wedge gcd(N)\chi \in Call(P, I) \wedge$$
$$(N >= M)\chi \in R^{SSE}_{rel(N>=M)} \wedge (M > 0)\chi \in R^{SSE}_{rel(M>0)} \wedge$$
$$(L \; is \; N - M)\chi \in R^{SSE}_{rel(L \; is \; N-M)} \rightarrow$$
$$[\![gcd(M), gcd(N)]\!]\chi \succ_\mu [\![C \mid C \in [\![gcd(M), gcd(L)]\!]\chi \wedge C \in Call(P, I)]\!].$$

The condition for the first rule is trivially satisfied. The condition for the second rule is also satisfied. That is, by considering that each of the built-in constraints, $N >= M$, $M > 0$ and $L \; is \; N - M$, simultaneously need to belong to the extended success set of their respective predicates, we have that $N\chi > L\chi$ and thus $gcd(N)\chi \succ gcd(L)\chi$. Thus, P is terminating for I.

2.4.2 Merge-sort

The following CHR program P is the merge-sort algorithm from Example 2.1.4, of which the use has been discussed in Example 2.2.1:

R_1 @ $msort([]) \Leftrightarrow true$.
R_2 @ $msort([L|Ls]) \Leftrightarrow r(0, L), msort(Ls)$.
R_3 @ $r(D, L1), r(D, L2) \Leftrightarrow less(L1, L2) \mid r(s(D), L1), a(L1, L2)$.
R_4 @ $a(L1, L2) \setminus a(L1, L3) \Leftrightarrow less(L2, L3) \mid a(L2, L3)$.

For P, the query set I and call set $Call(P, I)$ are discussed in Example 2.3.7. That is, I is the infinite set of $msort/1$ constraints, with at their argument position a list of 2^n ground terms of the form $s(s(\ldots s(0)))$. The call set $Call(P, I)$ is the set $\{msort(l),\ r(t_1, t_2),\ a(t_3, t_4),\ less(t_5, t_6) \mid l$ *is a bounded list of terms* t_0 *and* $t_0, t_1, \ldots, t_6$ *are terms of the form* $s(s(\ldots s(0)))\}$.

Note that R_4 is possibly non-terminating. Consider for example the application of R_4 on the constraints $a(s(0), s(0))$ and $a(s(0), s(s(0)))$, then the constraint $a(s(0), s(s(0)))$ is replaced by an identical constraint, allowing R_4 to fire in an identical way. This, however, can never be the case in the intended use of the program as only $a(L1, L2)$ constraints are added to the constraint store, by R_3 and R_4, such that $CT \models less(L1, L2)$.

In order to prove termination, therefore, we need to refine the description of the call set to reflect that the first argument of the $a/2$ constraints in the call set is of a strictly smaller term-size (see Definition 2.3.12) than the second as $CT \models less(t_1, t_2) \rightarrow |t_2|_{ts} > |t_1|_{ts}$. Thus, $Call(P, I) = \{msort(l),\ r(t_1, t_2),\ a(t_3, t_4),\ less(t_5, t_6) \mid l$ *is a bounded list of terms* t_0 *and* $t_0, t_1, \ldots, t_6$ *are terms of the form* $s(s(\ldots s(0)))$, *where* $|t_4|_{ts} > |t_3|_{ts}\}$. Note that in Section 6.5, we prove termination of this program automatically.

To prove termination of P for I, we construct on $Call(P, I)$ a reduction pair $(\succeq, \succ)$, with associated equivalence relation $\approx$, as follows:

- $\forall msort(x), msort(y) \in Call(P, I)$ such that $|x|_{ll} \geq |y|_{ll}$, where $|.|_{ll}$ is list-length (see Definition 2.3.11): $msort(x) \succeq msort(y)$.
- $\forall x, y, z \in Call(P, I)$ such that $rel(x)$ is $msort/1$, $rel(y)$ is $r/2$ and $rel(z)$ is $a/2$: $x \succ y$ and $x \succ z$.
- $\forall x, y \in Call(P, I)$ such that $rel(x)$ and $rel(y)$ are $r/2$: $x \approx y$,
- $\forall x, y \in Call(P, I)$ such that $rel(x)$ is $r/2$ and $rel(y)$ is $a/2$: $x \succ y$.
- $\forall a(w, x), a(y, z) \in Call(P, I)$ such that $|x|_{ts} - |w|_{ts} \geq |z|_{ts} - |y|_{ts}$, where $|.|_{ts}$ is term-size (see Definition 2.3.12): $a(w, x) \succeq a(y, z)$.

Furthermore, all built-in constraints of the call set are order equivalent and strictly smaller than any of the CHR constraints of the call set, such that they

can be ignored in the decrease conditions of the RC for abstract CHR. It can be verified that the call set is rigid w.r.t. $(\succeq, \succ)$.

Finally, to prove termination, we need the extended success set of the $less/2$ predicate: $R^{SSE}_{less/2} = \{less(a,b) \mid a\ is\ a\ term\ of\ the\ form\ s(s(\ldots s(0)))\ and\ b\ is\ a\ term\ of\ the\ form\ s(s(\ldots s(c)))\ with\ c \in \{0\} \cup Var_P,\ where\ |b|_{ts} > |a|_{ts}\}$.

Let $(\succeq_\mu, \succ_\mu)$ be the multiset extension of $(\succeq, \succ)$. Then, P for I satisfies the RC for abstract CHR if for any substitution χ of the variables of R_1 holds that

$$msort([])\chi \in Call(P,I) \rightarrow [\![msort([])]\!]\chi \succ_\mu [\![C \mid C \in [\![]\!]\chi \wedge C \in Call(P,I)]\!],$$

for any substitution χ of the variables of R_2 holds that

$$msort([L|Ls])\chi \in Call(P,I) \rightarrow$$

$$[\![msort([L|Ls])]\!]\chi \succ_\mu [\![C \mid C \in [\![r(0,L), msort(Ls)]\!]\chi \wedge C \in Call(P,I)]\!],$$

for any substitution χ of the variables of R_3 holds that

$$r(D,L1)\chi \in Call(P,I) \wedge r(D,L2)\chi \in Call(P,I) \wedge$$

$$less(L1,L2)\chi \in R^{SSE}_{rel(less(L1,L2))} \rightarrow$$

$$[\![r(D,L1), r(D,L2)]\!]\chi \succ_\mu [\![C \mid C \in [\![r(s(D),L1), a(L1,L2)]\!]\chi \wedge C \in Call(P,I)]\!]$$

and, finally, for any substitution χ of the variables of R_4 holds that

$$a(L1,L2)\chi \in Call(P,I) \wedge a(L1,L3)\chi \in Call(P,I) \wedge$$

$$less(L2,L3)\chi \in R^{SSE}_{rel(less(L2,L3))} \rightarrow$$

$$[\![a(L1,L2), a(L1,L3)]\!]\chi \succ_\mu [\![C \mid C \in [\![a(L1,L2), a(L2,L3)]\!]\chi \wedge C \in Call(P,I)]\!].$$

The first condition from above is trivially satisfied. For the second condition, for any χ, we have that $msort([L|Ls])\chi \succ msort(Ls)\chi$ because their arguments demonstrate a strict decrease in list-length: $|[L|Ls]\chi|_{ll} > |Ls\chi|_{ll}$. As furthermore any CHR constraint of the $r/2$ predicate is strictly smaller than any CHR constraint of the $msort/1$ predicate, the second condition is satisfied. For the third condition, for any χ, the number of $r/2$ constraints decreases, which are all order equivalent, while only adding new $a/2$ constraints, of which those in the call set are strictly smaller than any of the $r/2$ constraints in the call set. Thus, the condition for the third rule is satisfied. Finally, the condition on the fourth rule is satisfied as well. That is, if the precondition is satisfied, then $|L1\chi|_{ts} < |L2\chi|_{ts}$, $|L1\chi|_{ts} < |L3\chi|_{ts}$ and $|L2\chi|_{ts} < |L3\chi|_{ts}$. Therefore, $|L3\chi|_{ts} - |L1\chi|_{ts} > |L3\chi|_{ts} - |L2\chi|_{ts}$ and thus $a(L1,L3)\chi \succ a(L2,L3)\chi$. Thus, P is terminating for I.

Chapter 3

Termination of CHR with propagation

The main challenge in proving termination of the full CHR language —thus of CHR programs that may contain any kind of rule— is dealing with propagation. Because propagation rules only add constraints, a fire-once policy is required to guarantee that any combination of constraints can fire a propagation rule at most once, as such avoiding the trivial non-termination of propagation.

As was discussed in the Introduction, in the presence of propagation, an approach on the basis of proving state-size decreases cannot be achieved using the constraint store alone. To observe decreases, we would need to keep track of information regarding a fire-once policy. Two different approaches exist for implementing a fire-once policy at the level of CHR states. One approach keeps a *history* for tracking the combinations of constraints that have already fired a propagation rule [Frü98, Sch05], preventing as such multiple applications of a propagation rule on the same combination of constraints. Obviously, such a history can only increase in size when applying a propagation rule.

A second approach keeps a *forecast* on the combinations of constraints that still can fire a propagation rule [Abd97]. For certain applications of simplification rules, such a forecast may increase in size (see Chapter 4).

Therefore, in [VDSP08], a new approach was pursued. Instead of a termination argument based on state-size decreases, an argument is formulated guaranteeing *the finite addition of constraints* in any computation of the

program. In the *presence* of a fire-once policy, this implies program termination.

As has been shown in [PDS08b] (see Chapter 4), the technique of this chapter is less powerful than the technique based on state-size decreases of the next chapter. There is however an important motivation to discuss the approach of [VDSP08] in this text. That is, it is a novel approach, based on a new characterisation of CHR termination, that will prove to be useful in the correctness proofs of the next chapters.

Finally, note that, in contrast to [VDSP08], we consider the call set to be the set of constraints *used* by the rules of the program and that we extend the approach of [VDSP08] to general orderings.

3.1 The theoretical CHR semantics

In CHR, multiple occurrences of a CHR constraint can be present in a constraint store. Since all these occurrences, recombined with the same partner constraints, can fire the same propagation rule once, we need to be able to differentiate between multiple occurrences of a CHR constraint. Thus, constraints need labelling.

Definition 3.1.1 (labelled constraint)**.** *A* labelled constraint $c\sharp i$ *consists of a constraint* c *and a label* i*. We define* $con(c\sharp i) = c$ *to obtain the constraint and* $id(c\sharp i) = i$ *to obtain the label.* □

Although it is not required, note that we also label built-in constraints. It will allow for a more elegant formalisation of the theoretical CHR semantics and the termination conditions of this and the next chapter. This is because both the built-in and the CHR constraints of a CHR program are kept in a single constraint store, of which the elements are described by a single set $Atom_P$.

To label constraints, we use positive integer numbers. Therefore, to consider all labelled constraints constructible from the language $\mathcal{L}$ underlying P, we have to combine the elements of $Atom_P$ with labels in $\mathbb{N}$, and we define this set as $Atom_P^{\mathbb{N}} = \{c\sharp i \mid c \in Atom_P \wedge i \in \mathbb{N}\}$. Thus, the following equalities hold: $Atom_P = \{con(c) \mid c \in Atom_P^{\mathbb{N}}\}$ and $\mathbb{N} = \{id(c) \mid c \in Atom_P^{\mathbb{N}}\}$. Finally, we call a set S a *slice* of $Atom_P^{\mathbb{N}}$ iff $S \subseteq Atom_P^{\mathbb{N}} \wedge \forall c, c' \in S : id(c) \neq id(c')$.

Under the theoretical CHR semantics, we are no longer concerned with multisets of CHR constraints. Rules operate on sets of uniquely labelled constraints. However, note that if labels are omitted, a set of labelled CHR constraints can still correspond to a multiset of CHR constraints.

Definition 3.1.2 (constraint store)**.** *Let P be a CHR program. Then, S is called a* constraint store *for P iff S is a slice of $Atom_P^{\mathbb{N}}$.* □

As mentioned before, there are two approaches for implementing a fire-once policy: a *propagation history* [Sch05] for preventing multiple applications of propagation rules or a *propagation forecast* [Abd97] for allowing propagation rules to be fired on combinations of constraints. In either case, we represent this information using *tokens*.

Definition 3.1.3 (token)**.** *Let P be a CHR program and $\{h_1 \sharp id_1, \ldots, h_n \sharp id_n\}$ a slice of $Atom_P^{\mathbb{N}}$. Then, $(R, h_1 \sharp id_1, \ldots, h_n \sharp id_n)$ is called a* token *for P iff there exists a propagation rule*

$$R \mathbin{@} H_1, \ldots, H_n \Rightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

in P and $\exists \sigma\theta : CT \models ((h_1 = H_1) \wedge \cdots \wedge (h_n = H_n))\sigma \wedge (G_1 \wedge \cdots \wedge G_k)\sigma\theta$. By $Token_P$, we denote the set of tokens of a CHR program P. □

Note that in the setting of a propagation forecast, in order to verify whether a token corresponds to a *pending* application of a propagation rule, we consider unification and not matching. As such, also constraints that are not yet sufficiently instantiated to fire a propagation rule are accounted for. In the setting of a propagation history, we keep track of propagation rules that fired on sufficiently instantiated constraints, and thus can require matching instead.

The approach to termination analysis of CHR discussed in this chapter does not depend on how a fire-once policy is implemented. It is sufficient to consider its presence. Therefore, to define the theoretical semantics, we implement it here —in accordance to our needs for the next chapter (see Chapter 4)— by a propagation forecast.

Definition 3.1.4 (propagation forecast)**.** *Let P be a CHR program. Then, T is called a* propagation forecast *for P iff it is a subset of $Token_P$, representing pending applications of propagation rules in P.* □

Note that in [Abd97], a propagation forecast is called a *token store* instead. In our opinion, a token store should be regarded as the underlying data structure for a history or forecast, both differing only in the meaning they assign to the tokens in the token store. Thus, we define a *theoretical CHR state* as follows.

Definition 3.1.5 (theoretical CHR state)**.** *Let P be a CHR program. Then, a* theoretical CHR state *for P is an annotated tuple $\langle S, T \rangle_\nu$, where*

- *S is a constraint store for P,*

- *T is a propagation forecast for P and*
- *ν is a fresh label in $\mathbb{N}$ for the next constraint added to S.* □

Whenever adding a new CHR constraint D to the constraint store S, this new constraint can possibly fire a propagation rule with CHR constraints already present in the constraint store. Therefore, it can result in new pending applications $T^A_{(D,S)}$ of propagation rules.

Definition 3.1.6 (propagation forecast: addition of tokens)**.** *Let P be a CHR program, $\langle S, T\rangle_\nu$ a CHR state and D a labelled constraint added to the constraint store S, then*

$$\begin{aligned} T^A_{(D,S)} = \{&(R_p, h_1 \sharp i_1, \ldots, h_n \sharp i_n) \mid \\ &(R_p \ @\ H_1, \ldots, H_n \Rightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.) \in P, \\ &\{h_1 \sharp i_1, \ldots, h_n \sharp i_n\} \subseteq (\{D\} \cup S), D \in \{h_1 \sharp i_1, \ldots, h_n \sharp i_n\} \text{ and} \\ &\exists \sigma\theta : CT \models ((h_1 = H_1) \wedge \cdots \wedge (h_n = H_n))\sigma \wedge (G_1 \wedge \cdots \wedge G_k)\sigma\theta\}, \end{aligned}$$

where σ and θ are substitutions for unification and guard satisfaction, respectively. If multiple labelled constraints $D = \{D^1, \ldots, D^n\}$ are added, then

$$T^A_{(D,S)} = T^A_{(D^1,S)} \cup T^A_{(D^2,\{D^1\}\cup S)} \cup \cdots \cup T^A_{(D^n,\{D^1,\ldots,D^{n-1}\}\cup S)}.$$ □

Also, tokens $T^E_{(D,S)}$ in the propagation forecast T become invalid as a consequence of removing constraints D from the constraint store S. Contrary to [Abd97], we will not consider the elimination of tokens. This operation on the propagation forecast is not required from a semantical point of view (see Definition 3.1.7), nor will it be required to prove termination of a CHR program (see Section 4.3.3). This is because a token alone is not sufficient for rule application. That is, if the constraints of the token refer to constraints no longer available in the constraint store, the corresponding propagation rule cannot be applied using these constraints, as such invalidating the token.

Because the theoretical semantics does take a fire-once policy into account, a distinction is made between the three kinds of rules. However, since simplification can be regarded as a special case of simpagation, its semantics is not dealt with separately. That is, a simplification rule

$$H_1, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

can be written as a simpagation rule

$$true \setminus H_1, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

with an empty (*true*) kept head. The transition relation for CHR rules, under the theoretical CHR semantics, is therefore only defined in terms of simpagation and propagation.

Definition 3.1.7 (theoretical CHR transition relation)**.** *Let P be a CHR program and CT a constraint theory for the built-in constraints. Let θ represent the substitutions corresponding to the bindings generated when resolving built-in constraints by CT and let σ represent substitutions for the variables in the head of a rule as a result of matching. Then, the* transition relation $\rightarrow_P$ *for P, where $\langle S, T\rangle_\nu$ is a CHR state, is defined by:*

1. ***A solve transition**:*

   ***if** $S = \{b\sharp i_b\} \cup S'$, with b a built-in constraint, and $CT \models b\theta$,*
   ***then** $\langle S, T\rangle_\nu \rightarrow_P \langle S', T\rangle_\nu\theta$,*
   ***else if** $S = \{b\sharp i_b\} \cup S'$, $b \neq false$ and $\forall\theta : CT \not\models b\theta$,*
   ***or** $S = \{b\sharp i_b\} \cup S'$, $b = false$ and $S' \neq \emptyset$,*
   ***then** $\langle S, T\rangle_\nu \rightarrow_P \langle\{false\sharp\nu\}, T\rangle_{\nu+1}$.*

2. ***Propagation**:*

   ***if** $(R_p \ @\ H_1, \ldots, H_n \Rightarrow G_1, \ldots, G_k \mid D_1, \ldots, D_u.) \in P$,*
   ***and if** $S = \{h_1\sharp i_1, \ldots, h_n\sharp i_n\} \cup S'$,*
   $T = \{(R_p, h_1\sharp i_1, \ldots, h_n\sharp i_n)\} \cup T'$, *and*
   $CT \models (h_1 = H_1\sigma) \wedge \cdots \wedge (h_n = H_n\sigma) \wedge (G_1 \wedge \cdots \wedge G_k)\sigma\theta$,
   ***then** $\langle S, T\rangle_\nu \rightarrow_P \langle(\{D_1\sharp\nu, \ldots, D_u\sharp(\nu + u - 1)\} \cup S), T''\rangle_{\nu+u}\sigma\theta$,*
   ***where** $T'' = T' \cup T^A_{(\{D_1\sharp\nu,\ldots,D_u\sharp(\nu+u-1)\},S)}$.*

3. ***Simpagation**:*

   ***if** $(R_s \ @\ H_1, \ldots, H_j \setminus H_{j+1}, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid D_1, \ldots, D_u.) \in P$,*
   ***and if** $S = H_k \cup H_r \cup S'$,*
   $H_k = \{h_1\sharp i_1, \ldots, h_j\sharp i_j\}$,
   $H_r = \{h_{j+1}\sharp i_{j+1}, \ldots, h_n\sharp i_n\}$, *and*
   $CT \models (h_1 = H_1\sigma) \wedge \cdots \wedge (h_n = H_n\sigma) \wedge (G_1 \wedge \cdots \wedge G_k)\sigma\theta$,
   ***then** $\langle S, T\rangle_\nu \rightarrow_P \langle(\{D_1\sharp\nu, \ldots, D_u\sharp(\nu + u - 1)\} \cup H_k \cup S'), T'\rangle_{\nu+u}\sigma\theta$,*
   ***where** $T' = T \cup T^A_{(\{D_1\sharp\nu,\ldots,D_u\sharp(\nu+u-1)\},(H_k\cup S'))}$.*

Here, we represents by $[\![D_1, \ldots, D_u]\!]$ the added built-in and CHR constraints $[\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]$ in the body of a CHR rule. Similar to the abstract

CHR semantics, rules are non-deterministically applied until exhaustion. Which rule is applied, is a committed choice. Built-in constraints are assumed to return an answer in a finite number of steps and cannot introduce new constraints. If built-in constraints cannot be solved by the CT, the CHR program fails: $\langle\{false\sharp\nu\}, T\rangle_{\nu+1}$. □

We define the following kinds of CHR states.

Definition 3.1.8 (initial and final CHR states)**.** *Let P be a CHR program with transition relation $\rightarrow_P$. An* initial CHR state *or* query state *is a CHR state*

$$\langle\{c_1\sharp 1, \ldots, c_{(\nu-1)}\sharp(\nu-1)\}, T^A_{(\{c_1\sharp 1,\ldots,c_{(\nu-1)}\sharp(\nu-1)\},\{\})}\rangle_\nu,$$

where $\{c_1\sharp 1, \ldots, c_{(\nu-1)}\sharp(\nu-1)\}$ is an initial constraint store *for P. A* final CHR state *or* answer state *is a state Q, such that no CHR state Q' exists for which $(Q, Q') \in \rightarrow_P$. A final CHR state Q is a* failed CHR state *if $Q = \langle\{false\sharp\nu\}, T\rangle_{\nu+1}$, otherwise Q is a* successful CHR state. □

In the next example, we discuss a CHR program for computing prime numbers executed under the theoretical CHR semantics. Note that in computations, we may use a more concise denotation of tokens. That is, we only need to keep track of the labels of the constraints in the tokens and not of the labelled constraints themselves. This is allowed because tokens in CHR computations represent labelled constraints of the constraint store and since the latter is a slice of $Atom^{\mathbb{N}}_P$ there can be no confusion.

Example 3.1.1 (theoretical CHR semantics)**.** *The program below implements the Sieve of Eratosthenes for deriving the prime numbers up to a given number.*

$$R_1 \ @ \ primes(M) \setminus primes(N) \Leftrightarrow div(M, N) \mid true.$$
$$R_2 \ @ \ primes(N) \Rightarrow N > 2 \mid Np \ is \ N - 1, primes(Np).$$

R_2 generates the numbers for prime evaluation top-down. It adds, for every $primes(n)\sharp i$ constraint such that $CT \models n > 2$, the constraints Np is $n - 1\sharp\nu$ and $primes(Np)\sharp(\nu+1)$ once. *R_1 implements the sieve. For any two matching constraints $primes(m)\sharp i$ and $primes(n)\sharp j$, such that $m, n \in \mathbb{N}_0$ and m divides n, it removes $primes(n)\sharp j$.*

One possible initial CHR state for P is $\langle\{primes(6)\sharp 1\}, \{(R_2, 1)\}\rangle_2$, which can result in the following CHR computation (see Definition 2.3.2):

$$\begin{aligned}
\langle\{primes(6)\sharp 1\},\{(R_2,1)\}\rangle_2 &\xrightarrow{R_2}\\
\langle\{primes(6)\sharp 1, Np\ is\ 6-1\sharp 2, primes(Np)\sharp 3\},\{(R_2,3)\}\rangle_4 &\xrightarrow{CT}\\
\langle\{primes(6)\sharp 1, primes(5)\sharp 3\},\{(R_2,3)\}\rangle_4 &\xrightarrow{R_2}\\
\langle\{primes(6)\sharp 1, primes(5)\sharp 3, Np\ is\ 5-1\sharp 4, primes(Np)\sharp 5\},\{(R_2,5)\}\rangle_6 &\xrightarrow{CT}\\
\langle\{primes(6)\sharp 1, primes(5)\sharp 3, primes(4)\sharp 5\},\{(R_2,5)\}\rangle_6 &\xrightarrow{R_2}\\
\langle\{primes(6)\sharp 1, primes(5)\sharp 3, primes(4)\sharp 5, Np\ is\ 4-1\sharp 6, primes(Np)\sharp 7\},\{(R_2,7)\}\rangle_8 &\xrightarrow{CT}\\
\langle\{primes(6)\sharp 1, primes(5)\sharp 3, primes(4)\sharp 5, primes(3)\sharp 7\},\{(R_2,7)\}\rangle_8 &\xrightarrow{R_1}\\
\langle\{primes(5)\sharp 3, primes(4)\sharp 5, primes(3)\sharp 7\},\{(R_2,7)\}\rangle_8 &\xrightarrow{R_2}\\
\langle\{primes(5)\sharp 3, primes(4)\sharp 5, primes(3)\sharp 7, Np\ is\ 3-1\sharp 8, primes(Np)\sharp 9\},\{(R_2,9)\}\rangle_{10} &\xrightarrow{CT}\\
\langle\{primes(5)\sharp 3, primes(4)\sharp 5, primes(3)\sharp 7, primes(2)\sharp 9\},\{(R_2,9)\}\rangle_{10} &\xrightarrow{R_1}\\
\langle\{primes(5)\sharp 3, primes(3)\sharp 7, primes(2)\sharp 9\},\{(R_2,9)\}\rangle_{10} &\xrightarrow{\square}
\end{aligned}$$

Here, by $\xrightarrow{R_i}$ *we represent transitions of* $\rightarrow_P$ *due to the rules* R_i *in* P *and by* $\xrightarrow{CT}$ *transitions due to the CT. By* $\xrightarrow{\square}$ *we denote that no further transitions are possible, indicating that we have reached an answer state, which in this case is a successful CHR state. Consider in the above computation, the transition*

$$\begin{aligned}
\langle\{primes(6)\sharp 1, primes(5)\sharp 3\},\{(R_2,3)\}\rangle_4 &\xrightarrow{R_2}\\
\langle\{primes(6)\sharp 1, primes(5)\sharp 3, Np\ is\ 5-1\sharp 4, primes(Np)\sharp 5\},\{(R_2,5)\}\rangle_6 &
\end{aligned}$$

As $primes(Np)\sharp 5$ *can still get further instantiated such that the propagation rule* R_2 *is applicable on it, we consider a pending application of* R_2 *on* $primes(Np)\sharp 5$*. Thus, we add* $(R_2, 5)$ *to the propagation forecast. Note that* $(R_2, 3)$ *has been removed from the forecast since the transition is the result of applying* R_2 *on* $primes(5)\sharp 3$*. Consider now the transition*

$$\begin{aligned}
\langle\{primes(5)\sharp 3, primes(4)\sharp 5, primes(3)\sharp 7\},\{(R_2,7)\}\rangle_8 &\xrightarrow{R_2}\\
\langle\{primes(5)\sharp 3, primes(4)\sharp 5, primes(3)\sharp 7, Np\ is\ 3-1\sharp 8, primes(Np)\sharp 9\},\{(R_2,9)\}\rangle_{10} &
\end{aligned}$$

Even though $primes(Np)\sharp 9$*, at a later point in the computation, will get instantiated to* $primes(2)\sharp 9$ *on which* R_2 *is not applicable, we still consider the pending application of* R_2 *on* $primes(Np)\sharp 9$*, adding* $(R_2, 9)$ *to the propagation forecast. This is not a problem, since a token alone is not sufficient for the application of a propagation rule.* □

3.2 Termination of CHR with propagation

The termination conditions discussed in this chapter guarantee finite addition of constraints. For them to guarantee program termination, we prove that a program P with query set I terminates iff a finite number of constraints are added to the constraint store. First, however, we need to revisit the intended use of a CHR program and the success set of a CHR program.

3.2.1 The intended use and success set of a CHR program

For the theoretical CHR semantics, since constraints are labelled, we cannot reuse Definition 2.3.8 for the query set, Definition 2.3.9 for the call set and Definition 2.3.14 for the success set of a CHR program. So, we redefine them.

Definition 3.2.1 (query set)**.** *Let P be a CHR program and let $\{S_j \mid j \in J\}$ be a set of initial constraint stores of interest for P. Then, the set $I = \{c \mid c\sharp i \in \bigcup_{j\in J} S_j\}$ is called a* query set *for P.* □

Note that we do not extend the notion of a query set to a set of labelled constraints because, in general, we do not wish to reason on labels.

To relate initial theoretical CHR states to a query set I, we say that an initial CHR state $Q = \langle\{c_1\sharp 1, \ldots, c_{(\nu-1)}\sharp(\nu-1)\}, T^A_{(\{c_1\sharp 1,\ldots,c_{(\nu-1)}\sharp(\nu-1)\},\{\})}\rangle_\nu$ is *composed* from I iff for all $c_i\sharp i$, with $1 \leq i \leq (\nu-1)$, holds that $c_i \in I$.

From a query set I, we can derive a characterisation of the possible calls that can occur during computations of a program P for I. Contrary to [VDSP08], we consider the call set to be the set of constraints *used* to fire the rules of the program instead of the constraints *added* by the rules of the program.

Definition 3.2.2 (call set)**.** *Let P be a CHR program and I a query set for P. For any initial CHR state Q composed from I, the* call set *$Call(P, Q)$ w.r.t. Q is the closure under substitution of the set containing all built-in and CHR constraints C for which a variant of C is either resolved by CT or* used *to fire a CHR rule of P, in some computation of P for Q. Then, the call set $Call(P, I)$ w.r.t. I is the set $\bigcup_Q Call(P, Q)$, where Q is composed from I.* □

Recall that by considering the CHR constraints used by the CHR rules of the program, we may improve precision of the termination analysis by taking the success sets of added built-in constraints of the body of CHR rules into account, whereas [VDSP08] cannot (see Section 2.3.3 and Section 2.3.5).

Finally, we need to redefine the success set of a CHR program.

Definition 3.2.3 (success set of a CHR program)**.** *Let P be a CHR program. The success set R^{SS}_P of P is the set of all $A \in B_P$ such that there exists a CHR state $\langle\{A\sharp i\} \cup S, T\rangle_\nu$, with S possibly empty, that is an initial CHR state in a successful computation of P. The success set $R^{SS}_{c/n}$ of a predicate c/n is the set $\{A \mid A \in R^{SS}_P \wedge rel(A) = c/n\}$.* □

Note that we can reuse Proposition 2.3.3, if in the proof we consider theoretical CHR states instead of abstract CHR states. Furthermore, note that we can reuse Definition 2.3.15, defining extended success sets.

3.2.2 Termination of CHR with propagation

For proving termination by the finite addition of constraints, we only need to consider presence of a fire-once policy for propagation.

Lemma 3.2.1 (CHR termination)**.** *A CHR program P with query set I terminates iff there are a finite number of additions of constraints to the constraint store during any computation of P for Q, composed from I.* □

Proof. Let Q be any initial CHR state composed from a query set I.
$\Longrightarrow$: If P terminates for Q, then all computations of P for Q are finite (see Lemma 2.3.1) and thus only a finite number of rules are applied. Therefore, only a finite number of constraints are added to the store.
$\Longleftarrow$: Suppose there are only a finite number of constraints added to the store in any computation of a program P for Q. Each propagation rule can only be fired a finite number of times because of a fire-once policy. Therefore, there are only a finite number of transitions in the considered computation due to propagation. Each simpagation rule removes at least one CHR constraint from the constraint store. Therefore, there can exist only a finite number of transitions due to simpagation. Every solve transition removes a built-in constraint. Therefore, there can exist only a finite number of transitions due to solving built-in constraints. Since there are only a finite number of transitions in any computation of P for Q, P terminates for Q (see Lemma 2.3.1). Thus, P terminates for I. □

3.2.3 The RC for CHR with propagation

The termination conditions of the approach of this chapter were first discussed in [VPDS07] using total orderings. Afterwards, when automating the approach in [VDSP08], these were extended to partial orderings. Here, we discuss them in the context of pre-orderings.

To define the termination conditions for CHR with propagation, we first need to introduce the notion of a $(\succeq, \succ)$-*maximal multisubset*.

Definition 3.2.4 ($(\succeq, \succ)$-maximal multisubset)**.** *Let $M \uplus D$ be a multiset with universe U, on which a reduction pair $(\succeq, \succ)$ is defined. Then, D is a $(\succeq, \succ)$-maximal multisubset of $M \uplus D$ iff D is non-empty and such that*

- *for every two elements a and b in D holds that $a \approx b$ and*
- *given any element a in D, no element c in M exists for which $c \succeq a$.* □

Note that the notion of a $(\succeq, \succ)$-maximal multisubset is an instance of the notion of a $(\succeq, \succ)$-equivalent multisubset from the previous chapter (see Definition 2.3.6). That is, a $(\succeq, \succ)$-maximal multisubset is a $(\succeq, \succ)$-equivalent multisubset such that no other $(\succeq, \succ)$-equivalent multisubset exists with elements larger than the elements of the $(\succeq, \succ)$-maximal multisubset.

A multiset may have several different $(\succeq, \succ)$-maximal multisubsets. We illustrate this in the next example.

Example 3.2.1 ($(\succeq, \succ)$-maximal multisubset)**.** *Consider a reduction pair* $(\succeq, \succ)$ *for a universe* $U = \{a, b, c, d\}$*, where*

$$\succeq = \{(a,a), (b,b), (c,c), (d,d), (a,c), (b,d), (d,b)\}$$

is a pre-order relation with associated strict partial order relation $\succ = \{(a,c)\}$*. Then, a multiset* $M = [\![a, a, b, b, c, d, d]\!]$ *has the* $(\succeq, \succ)$*-maximal multisubsets:* $[\![a, a]\!]$ *and* $[\![b, b, d, d]\!]$*.* □

Informally, a program satisfies the RC for CHR with propagation if each propagation rule only adds constraints which are of a strictly smaller size than the constraints used in the head and if each simpagation rule reduces the number of largest sized constraints in the rule [VDSP08]. For pre-orderings, these conditions are generalised as follows.

Definition 3.2.5 (RC for CHR with propagation)**.** *Let* P *be a CHR program and* I *a query set. Let* $(\succeq, \succ)$ *be a reduction pair on* $Call(P, I)$*, such that* $Call(P, I)$ *is rigid w.r.t.* $(\succeq, \succ)$*. Then,* P *for* I *satisfies the* RC for CHR with propagation *iff:*

- *For any substitution* χ *of a propagation rule*

  $$R_p \ @ \ H_1, \ldots, H_n \Rightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

  in P*, such that*

  $$\forall H \in [\![H_1, \ldots, H_n]\!]\chi : H \in Call(P, I),$$
  $$\forall G \in [\![G_1, \ldots, G_k]\!]\chi : G \in R^{SSE}_{rel(G)}, \text{ and}$$
  $$\forall B \in [\![B_1, \ldots, B_l]\!]\chi : B \in R^{SSE}_{rel(B)},$$

  holds that $H \succ C$ *for all* $H \in [\![H_1, \ldots, H_n]\!]\chi$ *and all* $C \in [\![D \mid D \in [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi \wedge D \in Call(P, I)]\!]$*.*

- *For any substitution* χ *of a simpagation rule*

  $$R_s \ @ \ H_1, \ldots, H_j \setminus H_{j+1}, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

  in P*, such that*

$$\forall H \in [\![H_1, \ldots, H_n]\!]\chi : H \in Call(P, I),$$
$$\forall G \in [\![G_1, \ldots, G_k]\!]\chi : G \in R^{SSE}_{rel(G)}, \text{ and}$$
$$\forall B \in [\![B_1, \ldots, B_l]\!]\chi : B \in R^{SSE}_{rel(B)},$$

holds that for any $(\succeq, \succ)$*-maximal multisubset*

$$A_i = [\![H_{i_1}, \ldots, H_{i_p}, B_{i_1}, \ldots, B_{i_q}, C_{i_1}, \ldots, C_{i_r}]\!]$$

of $[\![D \mid D \in [\![H_{j+1}, \ldots, H_n, B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi \wedge D \in Call(P, I)]\!]$*, the cardinality decreases,*

$$\sharp([\![H_{i_1}, \ldots, H_{i_p}]\!]) > \sharp([\![B_{i_1}, \ldots, B_{i_q}, C_{i_1}, \ldots, C_{i_r}]\!]),$$

between head and body. □

Note that since simplification is a special case of simpagation, we did not include termination conditions for simplification rules. Furthermore, note that our condition on simpagation rules in Definition 3.2.5 is more strict than the corresponding termination condition for such rules in Definition 2.3.16, formulated on simplification rules. That is, the requirement for the number of maximally sized CHR constraints to decrease between removed head and body implies a decrease on the multiset extension of the pre-order.

The inverse is not true. Therefore, it is possible that programs without propagation can be proven terminating using the RC of Definition 2.3.16 while they cannot be proven terminating using the RC of Definition 3.2.5 (see Section 3.3.4). Unfortunately, a straightforward extension of the RC from Definition 2.3.16 with the condition on propagation rules of Definition 3.2.5 is incorrect.

Example 3.2.2 (counter example multiset decrease)**.** *A CHR program*

$$a(s(N)), a(N) \Leftrightarrow a(s(N)). \qquad a(s(N)) \Rightarrow a(N).$$

does not terminate for all possible queries $a(s(t))$*, with* t *an arbitrary term. Because the first rule adds a newly labelled variant of the constraint matching with the first head of the rule, the propagation rule is applicable on the newly added variant. Therefore, the partner constraint removed in the first rule can be added again through propagation. As a result, the first rule can fire in an identical way as before.*

The RC for CHR cannot be fulfilled on this program. That is, the simplification rule cannot be shown to decrease in size, unless the size of the constraints used in the second head are larger than those used in the first head. Under this assumption, the condition on the propagation rule cannot be satisfied.

Using the condition on simplification rules from the previous chapter, we can show a multiset decrease for the first rule, while satisfying the condition on propagation that all heads should have a larger size than all body constraints. So, a straightforward extension of Definition 2.3.16 is incorrect. □

Finally, by using a natural extension of $(\succeq, \succ)$ on $Call(P, I)$ to $Atom_P$ (see Definition 2.3.17), we can reformulate the RC for CHR with propagation.

Corollary 3.2.1. *Let P be a CHR program and I a query set. Let $(\succeq, \succ)$ be a reduction pair on $Call(P, I)$ such that $Call(P, I)$ is rigid w.r.t. $(\succeq, \succ)$ and let $(\succeq_P, \succ_P)$ be the natural extension of $(\succeq, \succ)$ to $Atom_P$. Then, P for I satisfies the* RC for CHR with propagation *w.r.t. $(\succeq, \succ)$ iff:*

- *For any substitution χ of a propagation rule*

  $$R_p \ @ \ H_1, \ldots, H_n \Rightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

  in P, such that

  $$\begin{aligned} &\forall H \in [\![H_1, \ldots, H_n]\!]\chi : H \in Call(P, I), \\ &\forall G \in [\![G_1, \ldots, G_k]\!]\chi : G \in R^{SSE}_{rel(G)}, \text{ and} \\ &\forall B \in [\![B_1, \ldots, B_l]\!]\chi : B \in R^{SSE}_{rel(B)}, \end{aligned}$$

  holds that

  $$\forall H \in [\![H_1, \ldots, H_n]\!]\chi \text{ and } \forall C \in [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi : H \succ_P C.$$

- *For any substitution χ of a simpagation rule*

  $$R_s \ @ \ H_1, \ldots, H_j \setminus H_{j+1}, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

  in P, such that

  $$\begin{aligned} &\forall H \in [\![H_1, \ldots, H_n]\!]\chi : H \in Call(P, I), \\ &\forall G \in [\![G_1, \ldots, G_k]\!]\chi : G \in R^{SSE}_{rel(G)}, \text{ and} \\ &\forall B \in [\![B_1, \ldots, B_l]\!]\chi : B \in R^{SSE}_{rel(B)}, \end{aligned}$$

  holds that for any $(\succeq_P, \succ_P)$-maximal multisubset

  $$A_i = [\![H_{i_1}, \ldots, H_{i_p}, B_{i_1}, \ldots, B_{i_q}, C_{i_1}, \ldots, C_{i_r}]\!]$$

  of $[\![H_{j+1}, \ldots, H_n, B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi$, the cardinality decreases,

  $$\sharp([\![H_{i_1}, \ldots, H_{i_p}]\!]) > \sharp([\![B_{i_1}, \ldots, B_{i_q}, C_{i_1}, \ldots, C_{i_r}]\!]),$$

  between head and body. □

Proof. The proof is a simple adaption of the proof of Corollary 2.3.1, considering the RC for CHR with propagation instead (see Definition 3.2.5). □

3.2.4 Correctness of the RC for CHR with propagation

Since constraints are labelled, we require an ordering on labelled constraints.

Definition 3.2.6 (supernatural extension)**.** *Let $S_1 \subseteq S_2$ be two sets of constraints. Let $(\succeq, \succ)$ be a reduction pair on S_1 and let $(\succeq', \succ')$ be the natural extension of $(\succeq, \succ)$ to S_2. Furthermore, let $S_3 = S_2^{\mathbb{N}} = \{c\sharp i \mid c \in S_2 \wedge i \in \mathbb{N}\}$ be the set of labelled constraints of S_2. Then, $(\succeq'', \succ'')$, with $\succ''$ the strict partial order associated to $\succeq''$, is the* supernatural extension *of $(\succeq, \succ)$ on S_1 to S_3 iff $\forall a, b \in S_2$ and $\forall a\sharp i, b\sharp j \in S_3$ holds that $a \succeq' b \leftrightarrow a\sharp i \succeq'' b\sharp j$.* □

We have the following property of supernatural extensions.

Proposition 3.2.1. *Let $S_1 \subseteq S_2$ be two sets of constraints and let $S_3 = S_2^{\mathbb{N}} = \{c\sharp i \mid c \in S_2 \wedge i \in \mathbb{N}\}$ be the set of labelled constraints of S_2. Furthermore, let $(\succeq, \succ)$ be a reduction pair on S_1 and $(\succeq'', \succ'')$ its supernatural extension to S_3. Then, $(\succeq'', \succ'')$ is a reduction pair.* □

Proof. Considering that $(\succeq', \succ')$, the natural extension of $(\succeq, \succ)$ on S_1 to S_2, is a reduction pair (see Proposition 2.3.4), $(\succeq'', \succ'')$ is a reduction pair. □

Thus, considering $(\succeq, \succ)$ on the constraints of the call set, we obtain a reduction pair on $Atom_P^{\mathbb{N}}$ by the supernatural extension $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ of $(\succeq, \succ)$.

The RC for CHR guarantees termination of the CHR program P w.r.t. the query set I. The following sufficiency theorem can therefore be expressed.

Theorem 3.2.1. *If a CHR program P satisfies the RC for CHR with propagation for a query set I, then P is terminating for I.* □

Note that correctness of Theorem 3.2.1 is implied by Theorem 4.3.1 of the next chapter (see Section 5.1). The proof below can therefore be skipped. It is however included since the approach of this chapter is new. Correctness proofs can therefore be interesting from a methodological point of view.

Proof. To prove termination of P for I, Lemma 3.2.1 shows that it is sufficient to prove that the total number of constraints added during any computation of P for an initial CHR state Q, composed from I, is finite.

If P for I satisfies the RC for CHR with propagation, it has been satisfied with a reduction pair $(\succeq, \succ)$ such that $Call(P, I)$ is rigid w.r.t. $(\succeq, \succ)$. Let $(\succeq_P, \succ_P)$ be the natural extension of $(\succeq, \succ)$ to $Atom_P$ and let $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ be the supernatural extension of $(\succeq, \succ)$ to $Atom_P^{\mathbb{N}}$, the atoms of P. We consider

$Ord = Atom_P^{\mathbb{N}} \backslash \approx_P^{\mathbb{N}}$, the set of all equivalence classes of $Atom_P^{\mathbb{N}}$ w.r.t. $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$, representing the orders of $Atom_P^{\mathbb{N}}$, and consider the reduction pair $(\succeq_P^{\mathbb{N}'}, \succ_P^{\mathbb{N}'})$ on $Atom_P^{\mathbb{N}} \backslash \approx_P^{\mathbb{N}}$ associated to $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$. Thus, $\succeq_P^{\mathbb{N}'}$ is a partial order on Ord.

To prove finite addition of constraints, we first show that given satisfaction of the RC for CHR with propagation (see Corollary 3.2.1), a set of elements in $Atom_P^{\mathbb{N}}$ with orders $\{[max_1], \ldots, [max_i], \ldots, [max_n]\}$ in Ord exists, such that all constraints that can ever enter the constraint store must have an order smaller or equal to some $[max_i]$ in the set.

Let Q be some initial CHR state composed from I, with constraint store S. Let $\{M_1, \ldots, M_n\}$ be the $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$-maximal multisubsets of S. Then, the set of orders, $\{[max_1], \ldots, [max_i], \ldots, [max_n]\} \subseteq Ord$, of the largest constraints in S is obtained by mapping some element max_i in each of the n $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$-maximal multisubsets of $\{M_1, \ldots, M_n\}$ to Ord.

Given satisfaction of the RC for CHR with propagation, CHR rules can only add constraints c, with an order $[c]$, strictly smaller than some constraint h, with an order $[h]$, used in the head. Therefore, no constraints of an order strictly greater than some $[max_i]$ in $\{[max_1], \ldots, [max_n]\}$ can ever be added.

Secondly, all constraints that can ever enter the constraint store during a computation of P for I correspond to a finite number of orders $\mathbb{O} \subseteq Ord$. That is, $\succ_P^{\mathbb{N}'}$ is well-founded, rules can only add a finite number of constraints and any call to the host language is universally terminating.

We will prove by induction on the elements of $\mathbb{O}$ that for each possible order, only a finite number of constraints of that order can enter the store. Important to note here is that the orderings of constraints that match the head of a rule —and thus make up the call set— cannot change anymore due to further instantiations, but that CHR constraints added to the constraint store still can by solving built-in constraints of the program. The latter is not a problem since the RC for CHR with propagation considers all substitutions of a rule.

For induction, we organise $\mathbb{O}$ as follows: $\mathbb{O} = \{o_1, \ldots, o_n, o_{n+1}, \ldots, o_{n+m}\}$, where all o_i, with $1 \leq i \leq n$, represent $[max_i]$ in $\{[max_1], \ldots, [max_n]\}$ and where all o_i, with $n+1 \leq i \leq n+m$, represent orders in $\mathbb{O} \backslash \{[max_1], \ldots, [max_n]\}$ in such a way that for all k, with $1 \leq k \leq n+m$, there cannot exist a j, with $1 \leq j < k$, such that $o_k \succ_P^{\mathbb{N}'} o_j$. Thus, different assignments of orders to symbols o_i in $\mathbb{O}$ are allowed as long as $j < k$ holds whenever $o_j \succ_P^{\mathbb{N}'} o_k$ holds.

- Base case: order o_i for $1 \leq i \leq n$. Due to the RC for CHR with propagation, propagation rules never add new constraints of order o_i. For simpagation rules, each time that a CHR constraint of order o_i is added

to the constraint store by such a rule, the total number of constraints with order o_i in the constraint store decreases. Thus, such additions can only occur a finite number of times. So, the total number a_{o_i} of constraints of order o_i that can ever enter the store is finite.

- Induction step: order o_j for $n+1 \leq j \leq n+m$. We assume that the induction hypothesis holds for $o_1, \ldots, o_{j-1}$ and prove that it holds for o_j. Let $a_{o_1}, \ldots, a_{o_{j-1}}$ be the total number of constraints added to the store during a computation of P for I, of orders $o_1, \ldots, o_{j-1}$ respectively.

  Obviously, the constraint store of any initial CHR state Q, composed from I, only contains a finite number of constraints of order o_j.

  Due to the RC for CHR with propagation, if a propagation rule adds a constraint of order o_j to the store, then the constraints matching the head of this rule all have orders strictly larger than o_j. Because the total numbers $a_{o_1}, \ldots, a_{o_{j-1}}$ of constraints is finite and thus also the total number of constraints of higher orders must be finite, a fire-once policy guarantees that only a finite number of constraints of order o_j are added to the store due to propagation rules.

  Again due to the RC for CHR with propagation, for every simpagation rule adding constraints of order o_j to the store, there exists an order o_p, where $o_p \succeq_P^{\mathbb{N}'} o_j$, such that the number of constraints of order o_p decreases and such that no constraint of order higher than o_p is added to the store. By considering the multiset extension of $(\succeq_P^{\mathbb{N}'}, \succ_P^{\mathbb{N}'})$, for every application of a simpagation rule, the multiset size of the multisubset of constraints with orders $\{o_1, \ldots, o_p\}$ in the constraint store decreases. Since this is a terminating process, only a finite number of constraints of order o_j can be added the store due to simpagation rules.

As resolving built-in constraints does not result in the addition of new constraints, we have proved that if the RC for CHR with propagation is satisfied, only a finite number of constraints are added to the constraint store and thus that, by Lemma 3.2.1, P terminates for I. □

3.3 Termination of typical CHR programs

In this section, we discuss termination of a number of typical CHR programs by application of the RC for CHR with propagation.

3.3.1 Merge-sort

The CHR program P,

$$R_1 \ @ \ msort([]) \Leftrightarrow true.$$
$$R_2 \ @ \ msort([L|Ls]) \Leftrightarrow r(0, L), msort(Ls).$$
$$R_3 \ @ \ r(D, L1), r(D, L2) \Leftrightarrow less(L1, L2) \mid r(s(D), L1), a(L1, L2).$$

is the Merge-sort program from the previous chapter (see Section 2.4.2), where we omitted the fourth rule for reasons of simplicity.

The intended use I, the call set $Call(P, I)$ and the extended success set $R^{SSE}_{less/2}$ of the $less/2$ predicate are all given in Section 2.4.2. Therefore, we do not repeat them here. Furthermore, we reuse the reduction pair $(\succeq, \succ)$, as defined in Section 2.4.2, for which $Call(P, I)$ is rigid.

To prove satisfaction of the RC for CHR with propagation for P with I, we verify for R_1 that the cardinality of the largest ordered constraints of the rule decreases between head and body. Since the body is empty, this is obviously the case. For R_2, for all χ where $msort(L)\chi \in Call(P, I)$, the body constraints are ordered strictly smaller than the head constraint. Thus, the RC for CHR with propagation on simpagation rules is satisfied for the second rule. Finally, for R_3, since any $r/2$ constraint of the call set is strictly larger than any $a/2$ constraint of the call set and the number of $r/2$ constraints decreases for every application of R_3, we have that the condition on simpagation rules is satisfied for the third rule as well. Thus, P is terminating for I.

3.3.2 Primes

The CHR program P for computing prime numbers (see Example 3.1.1) consists of a simpagation and a propagation rule:

$$R_1 \ @ \ primes(M) \setminus primes(N) \Leftrightarrow div(M, N) \mid true.$$
$$R_2 \ @ \ primes(N) \Rightarrow N > 2 \mid Np \ is \ N - 1, primes(Np).$$

If the intended use of P is characterised using a query set $I = \{primes(t) \mid t \in \mathbb{N}\}$, we can derive from it, a call set $Call(P, I) = I \cup \{t_1 > 2, t_3 \ is \ t_2 - 1 \mid t_1, t_2 \in \mathbb{N} \wedge t_3 \in Var_P\}$.

To prove termination of P for I, we order the $primes/1$ constraints of the call set by their argument values using the reduction pair $(\geq, >)$ on positive integer numbers. Furthermore, we consider all built-in constraints of the call

set order equivalent and strictly smaller than any of the $primes/1$ constraints of the call set, such that they can be ignored in the decrease conditions of the RC for CHR with propagation. It can be verified that $(\succeq, \succ)$ is a reduction pair and that $Call(P, I)$ is rigid w.r.t. $(\succeq, \succ)$. Finally, the success sets of the built-in constraints of P are $R^{SS}_{>/2} = \{N > M \mid N, M \in \mathbb{N} \wedge N > M\}$ and $R^{SS}_{is/2} = \{L \ is \ N - M \mid N, M, L \in \mathbb{N} \wedge L = N - M\}$. It can be verified that $R^{SSE}_{>/2} = R^{SS}_{>/2}$ and $R^{SSE}_{is/2} = R^{SS}_{is/2}$.

To prove satisfaction of the RC for CHR with propagation for P with I, for R_1, we verify that the cardinality of the largest ordered constraints of the rule decreases between head and body. Obviously, this is the case since the body of R_1 is empty. For R_2, we verify a strict decrease between the constraints used in the head and the constraints added in the body of the propagation rule. This is the case as for any χ, where $primes(N)\chi \in Call(P, I) \wedge (N > 2)\chi \in R^{SSE}_{rel(N>2)} \wedge (Np \ is \ N - 1)\chi \in R^{SSE}_{rel(Np \ is \ N-1)}$, holds that $primes(N)\chi \succ primes(Np)\chi$. Thus, P is terminating for I.

3.3.3 Fibonacci

The CHR program P below computes Fibonacci numbers. The first two rules resolve base cases, while the third rule adds $fib/2$ constraints.

$$R_1 \ @ \ fib(N, M) \Rightarrow N = 0 \mid M = 0.$$
$$R_2 \ @ \ fib(N, M) \Rightarrow N = s(0) \mid M = 1.$$
$$R_3 \ @ \ fib(s(s(N)), M_1), fib(s(N), M_2) \Rightarrow fib(N, M), M_1 \ is \ M_2 + M.$$

The program is terminating if queried with $fib(N, M)$ constraints, where N is a positive integer in successor notation. From the query set $I = \{fib(t_1, t_2) \mid t_1 \ is \ a \ term \ of \ the \ form \ s(s(\ldots s(0))) \ and \ t_2 \in \mathbb{N} \cup Var_P\}$, we derive the call set $Call(P, I) = I \cup \{t_1 = 0, t_2 = 1, t_5 \ is \ t_3 + t_4, t_6 = 0, t_7 = s(0) \mid t_1, t_2, t_3, t_4 \in \mathbb{N} \wedge t_5 \in \mathbb{N} \cup Var_P \wedge t_6, t_7 \ are \ terms \ of \ the \ form \ s(s(\ldots s(0)))\}$.

For proving termination, we construct a reduction pair $(\succeq, \succ)$, comparing $fib/2$ constraints on the basis of their first argument's term-size (see Definition 2.3.12). Considering the built-in constraints of the call set, we consider them all order equivalent and strictly smaller than the CHR constraints of the call set, such that we can ignore them in the decrease conditions of the RC for CHR with propagation. It can be verified that $Call(P, I)$ is rigid w.r.t. $(\succeq, \succ)$.

The first two rules, R_1 and R_2, trivially satisfy the RC for CHR with propagation as these rules only add built-in constraints. For R_3, we verify for any χ, where $fib(s(s(N)), M_1)\chi \in Call(P, I) \wedge fib(s(N), M_2)\chi \in Call(P, I)$,

that the CHR constraint $fib(N, M)\chi$ in the body of R_3 is strictly smaller than each of the head constraints $fib(s(s(N)), M_1)\chi$ and $fib(s(N), M_2)\chi$. Since this is obviously the case, the RC for CHR with propagation is satisfied for the third rule as well. Thus, P is terminating for I.

3.3.4 Problem classes

The following CHR program P belongs to a problem class that cannot be proven terminating using the RC for CHR with propagation (see Definition 3.2.5), but can be proven terminating using the RC for abstract CHR (see Definition 2.3.16).

$$R_1 \ @ \ a(s(N)), a(N), a(N) \Leftrightarrow a(s(N)), a(N).$$
$$R_2 \ @ \ a(s(N)) \Leftrightarrow a(N).$$

The reason for this is that in order for the second rule to satisfy the RC for CHR with propagation, we need to require that $a(s(t)) \succ a(t)$ for any term t. Consequently, the condition for the first rule cannot be satisfied. That is, we cannot show a decrease in cardinality between the head and body of the rule, considering the maximally ranked constraints of the rule. Using instead the RC for abstract CHR, we can show a multiset decrease on the first rule, given a multiset decrease for the second rule, and this on the basis of an ordering given by $a(s(t)) \succ a(t)$ for any term t. Thus, P is terminating for any query.

A second problem class is given by the following terminating CHR program P. It cannot be proven terminating using either the RC for CHR with propagation (see Definition 3.2.5) or the RC for abstract CHR (see Definition 2.3.16).

$$R_1 \ @ \ a(s(N)), a(N), a(N) \Leftrightarrow a(s(N)), a(N).$$
$$R_2 \ @ \ a(s(N)) \Leftrightarrow a(N).$$
$$R_3 \ @ \ a(s(s(N))), a(s(s(N))) \Rightarrow a(N).$$

Similar as before, because of the restriction imposed by the second rule, i.e., $a(s(t)) \succ a(t)$ for any term t, the RC for CHR with propagation cannot be satisfied for the first rule. Clearly, the program cannot be handled by the RC for abstract CHR since there is propagation.

In the next chapter, we develop a new RC for CHR able to prove termination of CHR programs like the ones above. As it turns out, the RC for CHR of the next chapter is strictly more powerful than the RC for CHR with propagation and the RC for abstract CHR (see Section 5.1).

Chapter 4

Termination of CHR

Although there is overlap in the CHR programs they can handle (see Section 2.4.2 and Section 3.3.1), the two approaches discussed in the previous chapters are complementary (see Section 3.3). The RC for CHR with propagation (see Definition 3.2.5) cannot handle some of the programs without propagation that the RC for abstract CHR can handle and the RC for abstract CHR (see Definition 2.3.16) cannot handle any of the programs with propagation. Especially this, but also the fact that there are interesting programs which cannot be handled by either one of these approaches (see Section 3.3.4), is a strong motivation to develop a new, more general, approach for proving termination of CHR programs [PDS08a].

4.1 Problem description

An important consideration, when developing a generalised approach, is obtaining a *CHR state representation* that can be seen to decrease for terminating CHR programs with propagation. However, devising such a CHR state representation is not straightforward. As discussed before, the constraint store alone cannot be used for this purpose. It only grows in size for propagation. Furthermore, the propagation history —most often used in practice [Sch05]— also only increases in size for propagation. During propagation, tokens are added to a propagation history and none are removed.

Considering a propagation forecast instead, the state representation $\langle S, T \rangle_\nu$, as

introduced in the previous chapter, does seem appropriate to observe decreases for propagation transitions. To see this, recall the CHR program for computing prime numbers of Example 3.1.1:

$$R_1 \ @ \ primes(M) \setminus primes(N) \Leftrightarrow div(M,N) \mid true.$$
$$R_2 \ @ \ primes(N) \Rightarrow N > 2, Np \ is \ N-1 \mid primes(Np).$$

For reasons of simplicity, we have moved the built-in constraint from the body of the second rule in Example 3.1.1 to the guard such that it is solved deterministically. Thus, an initial CHR state $\langle\{primes(6)\sharp 1\}, \{(R_2, primes(6)\sharp 1)\}\rangle_2$ for P can result in the following CHR computation:

$$\begin{aligned}
\langle\{primes(6)\sharp 1\}, \{(R_2, primes(6)\sharp 1)\}\rangle_2 &\xrightarrow{R_2} \\
\langle\{primes(6)\sharp 1, primes(5)\sharp 2\}, \{(R_2, primes(5)\sharp 2)\}\rangle_3 &\xrightarrow{R_2} \\
\langle\{primes(6)\sharp 1, primes(5)\sharp 2, primes(4)\sharp 3\}, \{(R_2, primes(4)\sharp 3)\}\rangle_4 &\xrightarrow{R_2} \\
\langle\{primes(6)\sharp 1, primes(5)\sharp 2, primes(4)\sharp 3, primes(3)\sharp 4\}, \{(R_2, primes(3)\sharp 4)\}\rangle_5 &\xrightarrow{R_1} \\
\langle\{primes(5)\sharp 2, primes(4)\sharp 3, primes(3)\sharp 4\}, \{(R_2, primes(3)\sharp 4)\}\rangle_5 &\xrightarrow{R_2} \\
\langle\{primes(5)\sharp 2, primes(4)\sharp 3, primes(3)\sharp 4, primes(2)\sharp 5\}, \{(R_2, primes(2)\sharp 5)\}\rangle_6 &\xrightarrow{R_1} \\
\langle\{primes(5)\sharp 2, primes(3)\sharp 4, primes(2)\sharp 5\}, \{(R_2, primes(2)\sharp 5)\}\rangle_6 &\xrightarrow{\square}
\end{aligned}$$

As in Example 3.1.1, by $\xrightarrow{R_i}$ we represent transitions of $\rightarrow_P$ due to the rules R_i in P. By $\xrightarrow{\square}$ we denote that no further transitions are possible, indicating that we have reached an answer state. In contrast to Example 3.1.1, we have made the labelled constraints of the tokens of the computation explicit.

If the intended use of P is given by a query set $I = \{primes(t) \mid t \in \mathbb{N}\}$, we can derive a call set $Call(P,I) = I \cup \{t_1 > 2, t_3 \ is \ t_2 - 1 \mid t_1, t_2 \in \mathbb{N} \wedge t_3 \in Var_P\}$. On $Call(P,I)$, we construct a reduction pair $(\succeq, \succ)$ by ordering the $primes/1$ constraints of the call set by their argument values using the reduction pair $(\geq, >)$ on positive integer numbers. Furthermore, we consider all built-in constraints of the call set order equivalent and strictly smaller than the $primes/1$ constraints of the call set. Then, $Call(P,I)$ is rigid w.r.t. $(\succeq, \succ)$.

To define an ordering on the tokens $Token_P = \{(R_2, primes(t)\sharp i) \mid t \in \mathbb{N} \cup Var_P \wedge i \in \mathbb{N}\}$ of the Primes example from above, we consider the constraints of the tokens and not their rule identifiers. Thus considering $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$, the supernatural extension of $(\succeq, \succ)$ to $Atom_P^{\mathbb{N}}$, we define $(R_2, c_i\sharp i) \succeq_\tau (R_2, c_j\sharp j)$ iff $c_i\sharp i \succeq_P^{\mathbb{N}} c_j\sharp j$. Associated to $\succeq_\tau$, we define the strict partial ordering $\succ_\tau$ in the usual way. As such, we obtain a reduction pair $(\succeq_\tau, \succ_\tau)$ on $Token_P$ (see Section 4.2 for the general approach to construct a reduction pair on tokens).

Finally, by $(\succeq_{\mu_P}^{\mathbb{N}}, \succ_{\mu_P}^{\mathbb{N}})$ we denote the multiset extension of $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ and by $(\succeq_{\mu_\tau}, \succ_{\mu_\tau})$ the multiset extension of $(\succeq_\tau, \succ_\tau)$.

Using these extensions, for any propagation transition, e.g.,

$$\langle\{primes(6)\sharp 1\},\{(R_2,primes(6)\sharp 1)\}\rangle_2 \xrightarrow{R_2} \langle\{primes(6)\sharp 1,primes(5)\sharp 2\},\{(R_2,primes(5)\sharp 2)\}\rangle_3,$$

the size of the propagation forecast decreases. For simpagation, e.g.,

$$\langle\{primes(6)\sharp 1,primes(5)\sharp 2,primes(4)\sharp 3,primes(3)\sharp 4\},\{(R_2,primes(3)\sharp 4)\}\rangle_5 \xrightarrow{R_1}$$
$$\langle\{primes(5)\sharp 2,primes(4)\sharp 3,primes(3)\sharp 4\},\{(R_2,primes(3)\sharp 4)\}\rangle_5,$$

the size of the propagation forecast remains constant, while the size of the constraint store decreases.

Thus, in the above computation, by combining the above reduction pairs for the constraint store and the propagation forecast into a lexicographical ordering on ordered pairs (T,S), a representation of CHR states $\langle S,T\rangle_\nu$, we can show strict decreases for the consecutive states of the computation.

Unfortunately, such behaviour is not always the case.

Example 4.1.1 (counter example)**.** *Recall the CHR program P,*

$$R_1 \ @\ a(s(N)),a(N),a(N) \Leftrightarrow a(s(N)),a(N).$$
$$R_2 \ @\ a(s(N)) \Leftrightarrow a(N).$$
$$R_3 \ @\ a(s(s(N))),a(s(s(N))) \Rightarrow a(N).$$

from Section 3.3.4. Consider an initial CHR state,

$$Q = \langle\{a(s(s(s(0))))\sharp 1,a(s(s(0)))\sharp 2,a(s(s(0)))\sharp 3,a(s(s(0)))\sharp 4\},$$
$$\{(R_3,2,3),(R_3,3,2),(R_3,2,4),(R_3,4,2),(R_3,3,4),(R_3,4,3)\}\rangle_5,$$

for which P is terminating. Recall that in CHR computations we can represent the labelled constraints in tokens by their labels only (see Section 3.1).

By exhaustively applying the propagation rule R_3 of P on the current state, we obtain the CHR state:

$$Q' = \langle\{a(s(s(s(0))))\sharp 1,a(s(s(0)))\sharp 2,a(s(s(0)))\sharp 3,a(s(s(0)))\sharp 4,$$
$$a(0)\sharp 5,a(0)\sharp 6,a(0)\sharp 7,a(0)\sharp 8,a(0)\sharp 9,a(0)\sharp 10\},\{\}\rangle_{11}.$$

Applying on Q' the simpagation rule R_1, yields a state:

$$Q'' = \langle\{a(s(s(0)))\sharp 4,a(0)\sharp 5,a(0)\sharp 6,a(0)\sharp 7,a(0)\sharp 8,a(0)\sharp 9,a(0)\sharp 10,$$
$$a(s(s(s(0))))\sharp 11,a(s(s(0)))\sharp 12\},\{(R_3,4,12),(R_3,12,4)\}\rangle_{13}.$$

As can be noticed for this simpagation transition, T does not remain constant. It increases in size. □

So, a lexicographical ordering on a state representation (T,S), as used in the Primes example from above, cannot be used to prove termination of the second problem from Section 3.3.4.

4.1.1 A new CHR state representation

Even though for simpagation in terminating CHR programs the size of the propagation forecast may increase, usually it can be shown to decrease for propagation. However, instead of (T, S), it should be clear that a lexicographical ordering on ordered pairs (S, T) cannot be used either.

> The key to solving this problem *is to reconsider the meaning of propagation. Declaratively, propagation corresponds to logical implication and therefore can be regarded as a way to "complete" a state's information content, making explicit what is implied by the current state. When comparing CHR states, we can therefore take the* effect of "full" propagation *on the current state into account, i.e., the constraints added as a consequence of exhaustively applying the propagation rules of a CHR program on the current state. As such, we can compare the "actual" information content of CHR states, rather than their content in "compressed" form.*

Based on this understanding, we consider a third component for our state representation: the *propagation store*; representing the effect of *full propagation* on the current state. It is the multiset of labelled constraints that can be added to the constraint store by propagation on the current state (See Definition 4.3.1). As such, we can construct a *new CHR state representation*, based on a lexicographical ordering, that can be seen to decrease for terminating CHR programs, such as the programs of Section 3.3.4. That is, for propagation in terminating CHR programs, the combined size of the propagation store and the constraint store cannot increase, while the size of the propagation forecast can be seen to decrease. For simpagation in terminating CHR programs, the combined size of propagation store and constraint store can be seen to decrease, while the propagation forecast remains finite.

We illustrate this first on the Primes computation from above (see Section 4.1). For each of the CHR states of the computation, we represent below the union of the constraint store and the propagation store:

$$
\begin{aligned}
\{primes(6)\sharp 1\} \cup \{primes(5)\sharp 2, primes(4)\sharp 3, primes(3)\sharp 4, primes(2)\sharp 5\} &\xrightarrow{R_2} \\
\{primes(6)\sharp 1, primes(5)\sharp 2\} \cup \{primes(4)\sharp 3, primes(3)\sharp 4, primes(2)\sharp 5\} &\xrightarrow{R_2} \\
\{primes(6)\sharp 1, primes(5)\sharp 2, primes(4)\sharp 3\} \cup \{primes(3)\sharp 4, primes(2)\sharp 5\} &\xrightarrow{R_2} \\
\{primes(6)\sharp 1, primes(5)\sharp 2, primes(4)\sharp 3, primes(3)\sharp 4\} \cup \{primes(2)\sharp 5\} &\xrightarrow{R_1} \\
\{primes(5)\sharp 2, primes(4)\sharp 3, primes(3)\sharp 4\} \cup \{primes(2)\sharp 5\} &\xrightarrow{R_2} \\
\{primes(5)\sharp 2, primes(4)\sharp 3, primes(3)\sharp 4, primes(2)\sharp 5\} &\xrightarrow{R_1} \\
\{primes(5)\sharp 2, primes(3)\sharp 4, primes(2)\sharp 5\} &\xrightarrow{\square}
\end{aligned}
$$

As can be verified, for propagation, the combined size of constraint and propagation store remains constant. For simpagation their combined size decreases. The same principle also applies to the counter example from above.

Example 4.1.2 (motivating example)**.** *Recall the state*

$$Q' = \langle \{a(s(s(s(0))))\sharp 1, a(s(s(0)))\sharp 2, a(s(s(0)))\sharp 3, a(s(s(0)))\sharp 4,$$

$$a(0)\sharp 5, a(0)\sharp 6, a(0)\sharp 7, a(0)\sharp 8, a(0)\sharp 9, a(0)\sharp 10\}, \{\}\rangle_{11}.$$

obtained in Example 4.1.1 by exhaustively applying R_3 of P on Q. Its propagation store is obviously empty since the token store is empty. Let S be a multiset of constraints, then the combined constraint and propagation stores can be represented as $[\![a(s(s(s(0)))), a(s(s(0)))]\!] \uplus S$, where we have omitted the labels of the labelled constraints.

Applying on Q' the simpagation rule R_1, yielded a state:

$$Q'' = \langle \{a(s(s(0)))\sharp 4, a(0)\sharp 5, a(0)\sharp 6, a(0)\sharp 7, a(0)\sharp 8, a(0)\sharp 9, a(0)\sharp 10,$$

$$a(s(s(s(0))))\sharp 11, a(s(s(0)))\sharp 12\}, \{(R_3, 4, 12), (R_3, 12, 4)\}\rangle_{13}.$$

Its propagation store contains two constraints $a(0)$ representing the effect of full propagation. The combined constraint and propagation stores can therefore be represented as $[\![a(s(s(s(0)))), a(0), a(0)]\!] \uplus S$.

Thus, for the considered simpagation transition, T remains finite, while the combined constraint and propagation stores can be seen to decrease in size. □

To avoid misunderstanding, note that the introduction of the propagation store does not mean that we restrict to computation rules in which simpagation is interleaved with full propagation. We will still allow propagation to be interrupted by simpagation. So, there is no restriction on the computation rule. However, if we do interrupt full propagation by applying a simpagation rule, this affects the propagation store. That is, if constraints are added to the constraint store, new tokens are added to the propagation forecast. As a result, new constraints are also added to the propagation store. If constraints are removed from the constraint store, some tokens may become invalid. As a consequence, constraints in the propagation store are removed.

Furthermore, note that we will not explicitly define the added and removed constraints of the propagation store due to a simpagation transition. They are not needed in the termination conditions that we will propose. Moreover, the propagation store is redundant anyway. So, the new propagation store can always be determined by computing full propagation on a newly obtained state.

4.2 Termination of propagation

To characterise the propagation store, we need to distinguish between two different kinds of CHR states in CHR computations.

Definition 4.2.1 (full propagation)**.** *A* fully propagated state *is a state on which no solve and no propagation transitions are possible. A* partially propagated state *is a state on which solve or propagation transitions are still possible. By the action of* full (partial) propagation *on a state Q, we refer to sequences of solve and propagation transitions in CHR computations, starting in Q and ending in a fully (partially) propagated CHR state.* □

Note that there is good reason for considering solve transitions in our notion of full propagation. That is, a propagation rule can become applicable due to the solving of a built-in constraint. Therefore, if we would not consider solve transitions as part of full propagation sequences, it could be that a solve transition results in an increase of the propagation store. As a result, to verify decreases on the combined constraint and propagation stores, under assumption of a terminating host language, we would not be able to disregard built-in constraints in our termination conditions. More specifically, to observe a decrease on the combined constraint and propagation stores, the built-in constraint that is solved, and thus removed, must be assigned a strictly greater size than any of the constraints added to the propagation store as a result of solving the built-in constraint and it is exactly this what we wish to avoid.

Furthermore, note that full propagation may correspond to an infinite sequence of solve and propagation transitions. Consider for example a CHR program with a propagation rule $(a(N) \Rightarrow a(M).)$. Then, if the constraint store contains a constraint $a(t)\sharp i$, with t an arbitrary term, the effect of full propagation is infinite. Therefore, to guarantee termination, we will first guarantee finiteness of full propagation sequences and call this property *propagation safeness.*

Definition 4.2.2 (propagation safeness)**.** *A CHR program P is* propagation safe *for a query set I iff there are no infinite full propagation sequences in computations of P for I.* □

Thus, a propagation safe CHR program without simpagation is terminating.

Theorem 4.2.1. *Let P be a CHR program without simpagation. Then, P is propagation safe for a query set I iff P is terminating for I.* □

Proof. A direct consequence of Proposition 4.2.2 and Lemma 2.3.1. □

To prove propagation safeness, we will consider the constraint store for solve transitions, similar to Section 2.3.7. For propagation transitions, we will consider the propagation forecast instead.

4.2.1 The RC for CHR on propagation rules

Recall the RC on CHR with propagation from the previous chapter (see Definition 3.2.5). The condition on propagation rules guarantees that no constraint is ever added by a propagation rule, that through further propagation can replace one of the head constraints that gave cause to it. Under a fire-once policy, this corresponds intuitively to the notion of termination of propagation and, as it turns out, is sufficient to guarantee propagation safeness.

Definition 4.2.3 (RC for CHR on propagation rules)**.** *Let P be a CHR program and I a query set. Let $(\succeq, \succ)$ be a reduction pair on $Call(P, I)$, such that $Call(P, I)$ is rigid w.r.t. $(\succeq, \succ)$. Then, a CHR program P for I satisfies the* RC for CHR on propagation rules *iff:*

- *For any substitution χ of a propagation rule*

$$R_p \ @ \ H_1, \ldots, H_n \Rightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

in P, such that

$$\forall H \in [\![H_1, \ldots, H_n]\!]\chi : H \in Call(P, I),$$
$$\forall G \in [\![G_1, \ldots, G_k]\!]\chi : G \in R^{SSE}_{rel(G)}, \text{ and}$$
$$\forall B \in [\![B_1, \ldots, B_l]\!]\chi : B \in R^{SSE}_{rel(B)},$$

holds that $H \succ C$ for all $H \in [\![H_1, \ldots, H_n]\!]\chi$ and all $C \in [\![D \mid D \in [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi \wedge D \in Call(P, I)]\!]$. □

Note that the discussion of Section 3.2.3 is also relevant to this section. In particular, the result of Corollary 3.2.1 w.r.t. propagation rules is relevant to the proofs that follow.

4.2.2 Correctness of the RC for CHR on propagation rules

To prove propagation safeness using the RC for CHR on propagation rules, we construct a reduction pair on the tokens of the propagation forecast. Then by considering its multiset extension, we can verify strict order decreases of the propagation forecast for propagation transitions.

To construct a reduction pair on tokens, we first consider that in the RC for CHR on propagation rules it is required that any constraint used in the head of a propagation rule is of a strictly larger size than the constraints added in the body of the rule. Regarding the token removed in a propagation transition and the tokens added by a propagation transition, therefore, the RC guarantees existence of a constraint in each of the added tokens which is strictly smaller than all of the constraints in the removed token. Ordering tokens accordingly, it turns out that we obtain a well-founded ordering suitable for proving termination.

A second consideration regarding an ordering on tokens is that rigidity of the call set should be reflected at the level of the tokens. That is, once a token contains only constraints part of the call set of the program, its size may no longer change for further instantiations.

Definition 4.2.4 (token relation)**.** *Let P be a CHR program and $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ a reduction pair on constraints of $Atom_P^{\mathbb{N}}$. Let $t = (R_p, c_1 \sharp i_1, \ldots, c_j \sharp i_j, \ldots, c_n \sharp i_n)$ and $t' = (R_{p'}, c'_1 \sharp i'_1, \ldots, c'_{j'} \sharp i'_{j'}, \ldots, c'_{n'} \sharp i'_{n'})$ be tokens in $Token_P$. Then, the* token relation $\succeq_\tau$ *is defined as follows:*

1. $t \succ_\tau t'$ *iff there is a* $c'_{j'} \sharp i'_{j'}$ *such that for all* $c_j \sharp i_j$ *holds that* $c_j \sharp i_j \succ_P^{\mathbb{N}} c'_j \sharp i'_j$.
2. $t \approx_\tau t'$ *iff* $R_p = R_{p'}$ *and for all* j *holds that* $c_j \sharp i_j \approx_P^{\mathbb{N}} c'_j \sharp i'_j$ *or if there exists a* c_j *of* t *not in* $Call(P, I)$ *and a* $c_{j'}$ *of* t' *not in* $Call(P, I)$.
3. $t \succeq_\tau t'$ *iff* $t \succ_\tau t'$ *or* $t \approx_\tau t'$. □

Some discussion is in order. The first rule yields a strict partial ordering on tokens. As can be verified, the relation induced is transitive if the underlying ordering on constraints is transitive. The first part of the second rule guarantees reflexive closure. That is, for any two tokens of the same propagation rule, if all of the corresponding constraints are of the same size, then the tokens are of the same size. The second part of the second rule orders all tokens with constraints not in the call set equally. This will be more favourable in regard to Lemma 4.2.1. Finally, the third rule defines a pre-order relation based on the strict partial order relation and the equivalence relation.

Note that by the first rule, if the reduction pair on constraints of $Atom_P^{\mathbb{N}}$ is obtained as the supernatural extension of a reduction pair on constraints of the call set, we order tokens with constraints not in the call set strictly smaller than tokens with only constraints of the call set. Finally, note that by the first part of the second rule, in the case of a supernatural extension, if we have rigidity of the call set w.r.t. the selected reduction pair on constraints of the call set, that tokens with only constraints of the call set do not further change in size for further instantiations.

The token relation $\succeq_\tau$, based on a reduction pair $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ on labelled constraints, yields a reduction pair $(\succeq_\tau, \succ_\tau)$ on tokens.

Proposition 4.2.1. *Let P be a CHR program, $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ a reduction pair on labelled constraints of $Atom_P^{\mathbb{N}}$ and $\succeq_\tau$ a token relation based on $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ for the tokens in $Token_P$. Then, $(\succeq_\tau, \succ_\tau)$ is a reduction pair.* □

Proof. As can be verified, $\succeq_\tau$ is a pre-order. It is reflexive and transitive. Furthermore, $\succ_\tau$ is well-founded on $Token_P$. Otherwise, the existence of an infinite strict decreasing chain of tokens w.r.t. $\succ_\tau$ requires the existence of an infinite strict decreasing chain of constraints w.r.t. $\succ_P^{\mathbb{N}}$ and this would contradict the well-foundedness of $\succ_P^{\mathbb{N}}$. □

Note that the constraint store (see Definition 3.1.2) and the propagation forecast (see Definition 3.1.4) are sets. In the remainder of this chapter, we will consider them to be multisets. Therefore, without loss of generality, we will consider in Definition 3.1.6 and Definition 3.1.7 of the theoretical CHR semantics multiset join instead of set union and multisubsets instead of subsets.

We are now ready to prove correctness of the RC for CHR on propagation rules. That is, satisfaction of the RC for CHR on propagation rules guarantees propagation safeness. First, we introduce a new CHR state representation, similar to the abstract CHR state representation of Definition 2.3.18.

Definition 4.2.5 (CHR state representation for propagation)**.** *Let P be a CHR program and I a query set. Let $Q_0 \xrightarrow{P,\phi_1} Q_1 \xrightarrow{P,\phi_2} Q_2 \xrightarrow{P,\phi_3} \ldots \xrightarrow{P,\phi_q} Q_q \xrightarrow{P,\phi_{q+1}} \ldots$ be any computation of P for I, where $\phi_i = \theta_i$ —the answer substitution for the built-in constraint— if $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ is a solve transition and $\phi_i = \sigma_i\theta_i$ —the composition of match and answer substitution— if $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ is a propagation or a simpagation transition. Then, for $q \geq i$, we define*

- $\phi_i^q = \phi^\emptyset\phi_{i+1}\phi_{i+2}\ldots\phi_q$, *with $\phi^\emptyset$ the empty substitution, and*
- $|Q_i|_\pi^q = |\langle S_i, T_i\rangle_{\nu_i}|_\pi^q = (T_i, S_i)\phi_i^q$. □

Note that the CHR state representation from above cannot be used in general as discussed in Section 4.1. It is however useful to prove propagation safeness. We define the following order relation on the CHR state representation.

Definition 4.2.6 (CHR state ordering for termination of propagation)**.** *Let $(\succeq_{\mu_P}^{\mathbb{N}}, \succ_{\mu_P}^{\mathbb{N}})$ be the multiset extension of a reduction pair $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ on $Atom_P^{\mathbb{N}}$*

and $(\succeq_{\mu_\tau}, \succ_{\mu_\tau})$ *the multiset extension of a reduction pair* $(\succeq_\tau, \succ_\tau)$ *on* $Token_P$*, based on* $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$*. Then,* $\succeq_\pi$ *is the lexicographical ordering on ordered pairs* (T, S) *of the CHR state representation, where the first element is compared on the basis of* $(\succeq_{\mu_\tau}, \succ_{\mu_\tau})$ *and the second on the basis of* $(\succeq_{\mu_P}^{\mathbb{N}}, \succ_{\mu_P}^{\mathbb{N}})$*.* □

The CHR state ordering for propagation yields a reduction pair $(\succeq_\pi, \succ_\pi)$.

Proposition 4.2.2 (CHR state reduction pair for termination of propagation)**.** *Let* $(\succeq_{\mu_P}^{\mathbb{N}}, \succ_{\mu_P}^{\mathbb{N}})$ *be the multiset extension of a reduction pair* $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ *on* $Atom_P^{\mathbb{N}}$ *and* $(\succeq_{\mu_\tau}, \succ_{\mu_\tau})$ *the multiset extension of a reduction pair* $(\succeq_\tau, \succ_\tau)$ *on* $Token_P$*, based on* $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$*. Finally, let* $\succeq_\pi$ *be the CHR state ordering based on these extensions. Then,* $(\succeq_\pi, \succ_\pi)$ *is a reduction pair, where* $\succ_\pi$ *is the strict partial order associated to* $\succeq_\pi$*.* □

Proof. It is well-known that a lexicographical ordering, based on reduction pairs, yields a reduction pair (see e.g., [DM79] and [Der82]). □

We will also need the following lemma.

Lemma 4.2.1 (boundedness of $|.|_\pi^q$ in q)**.** *Let* P *be a CHR program and* I *a query set. Let* $Q_0 \xrightarrow{P,\phi_1} Q_1 \xrightarrow{P,\phi_2} Q_2 \xrightarrow{P,\phi_3} \ldots \xrightarrow{P,\phi_q} Q_q \xrightarrow{P,\phi_{q+1}} \ldots$ *be any computation of* P *for* I*. Let* $(\succeq, \succ)$ *be a reduction pair on* $Call(P, I)$ *for which* $Call(P, I)$ *is rigid. Let* $(\succeq_{\mu_P}^{\mathbb{N}}, \succ_{\mu_P}^{\mathbb{N}})$ *be the multiset extension of a reduction pair* $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ *on* $Atom_P^{\mathbb{N}}$*, obtained as the supernatural extension of* $(\succeq, \succ)$*, and* $(\succeq_{\mu_\tau}, \succ_{\mu_\tau})$ *the multiset extension of a reduction pair* $(\succeq_\tau, \succ_\tau)$ *on* $Token_P$*, based on* $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$*. Finally, let* $(\succeq_\pi, \succ_\pi)$ *be the CHR state reduction pair for propagation based on these extensions. Then, for any* Q_i *in the computation, there exists a* $q \geq i$*, with* Q_q *in the computation, such that for all* $q' \geq q$*, with* $Q_{q'}$ *in the computation, holds that* $|Q_i|_\pi^q \approx_\pi |Q_i|_\pi^{q'}$*.* □

Proof. $Q_i = \langle S, T \rangle_\nu$ is a finite state, where S is a slice of $Atom_P^{\mathbb{N}}$ and T a subset of $Token_P$. Then, considering rigidity of the call set, the proof is similar to the proof of Lemma 2.3.2. That is, for any $A \in S$, either for all $q \geq i$: $A\phi_i^q \approx_P^{\mathbb{N}} A$ or there exists a $q_A \geq i$ such that $A\phi_i^{q_A} \succ_P^{\mathbb{N}} A$ and for all $q' \geq q_A$: $A\phi_i^{q'} \approx_P^{\mathbb{N}} A\phi_i^{q_A}$. For any $B \in T$, either for all $q \geq i$: $B\phi_i^q \approx_\tau B$ or there exists a $q_B \geq i$ such that $B\phi_i^{q_B} \succ_\tau B$ and for all $q'' \geq q_B$: $B\phi_i^{q''} \approx_\tau B\phi_i^{q_B}$. Since both S and T are finite, by taking the maximum of all such q_A and q_B, we obtain the result. □

Using the CHR state representation from above, under satisfaction of the RC for CHR on propagation rules, we can prove that for any solve or propagation transition in a non-failed computation, there exists a strict decrease between the consecutive program states. We have the following lemma.

Lemma 4.2.2. *Let P be a CHR program that satisfies the RC for CHR on propagation rules for a query set I w.r.t. $(\succeq, \succ)$ on $Call(P, I)$. Let $(\succeq^{\mathbb{N}}_{\mu_P}, \succ^{\mathbb{N}}_{\mu_P})$ be the multiset extension of $(\succeq^{\mathbb{N}}_{P}, \succ^{\mathbb{N}}_{P})$ on $Atom^{\mathbb{N}}_{P}$, obtained as the supernatural extension of $(\succeq, \succ)$, and $(\succeq_{\mu_\tau}, \succ_{\mu_\tau})$ the multiset extension of a reduction pair $(\succeq_\tau, \succ_\tau)$ on $Token_P$, based on $(\succeq^{\mathbb{N}}_{P}, \succ^{\mathbb{N}}_{P})$. Finally, let $(\succeq_\pi, \succ_\pi)$ be the CHR state reduction pair for propagation, based on these extensions. Then, for any solve or propagation transition $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ in a non-failed computation $Q_0 \xrightarrow{P,\phi_1} Q_1 \xrightarrow{P,\phi_2} Q_2 \xrightarrow{P,\phi_3} \ldots \xrightarrow{P,\phi_q} Q_q \xrightarrow{P,\phi_{q+1}} \ldots$ of P for I, holds that $\exists q \geq i$ such that $\forall q' \geq q : |Q_{i-1}|^{q'}_\pi \approx_\pi |Q_{i-1}|^{q}_\pi \succ_\pi |Q_i|^{q}_\pi \approx_\pi |Q_i|^{q'}_\pi$.* □

Proof. There are two kinds of transitions to consider:

1. Let $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ be a solve transition in a non-failed computation of P for I and let Q_q be in the computation, with $q \geq i$. Then, necessarily,

   - $|Q_{i-1}|^q_\pi = (T, [\![b \sharp i_b]\!] \uplus S)\phi^q_{i-1}$,
   - $|Q_i|^q_\pi = (T, S)\theta_i\phi^q_i$, and
   - $\phi^q_{i-1} = \theta_i\phi^q_i$.

   The first component of the CHR state representation for propagation remains constant: $T\phi^q_{i-1} \approx_{\mu_\tau} T\theta_i\phi^q_i$ for all $q \geq i$. For the second component, as $[\![b \sharp i_b]\!]\phi^q_{i-1} \succ^{\mathbb{N}}_{\mu_P} [\![]\!]$, by constructiveness, $[\![b \sharp i_b]\!]\phi^q_{i-1} \uplus S\phi^q_{i-1} \succ^{\mathbb{N}}_{\mu_P} S\phi^q_{i-1}$. Thus the second component decreases in size. Therefore, $|Q_{i-1}|^q_\pi \succ_\pi |Q_i|^q_\pi$.

   Since this holds for any $q \geq i$, and by Lemma 4.2.1 there exists a q_{i-1}, such that for all $q' \geq q_{i-1}$, $|Q_{i-1}|^{q'}_\pi \approx_\pi |Q_{i-1}|^{q_{i-1}}_\pi$, and a q_i, such that for all $q' \geq q_i$, $|Q_i|^{q'}_\pi \approx_\pi |Q_i|^{q_i}_\pi$, we have that for $q = max\{q_{i-1}, q_i\}$, for all $q' \geq q : |Q_{i-1}|^{q'}_\pi \approx_\pi |Q_{i-1}|^{q}_\pi \succ_\pi |Q_i|^{q}_\pi \approx_\pi |Q_i|^{q'}_\pi$.

2. Let $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ be a propagation transition in a non-failed computation of P for I. Let $(R \ @ \ H_1, \ldots, H_n \Rightarrow G_1, \ldots, G_k \mid D_1, \ldots, D_u.)$ be the propagation rule of P applied in the transition and this on the CHR constraints $h_1, \ldots, h_n$ of Q_{i-1} with match substitution σ_i, such that the guards $G_1, \ldots, G_k$ all succeed with answer substitution θ_i. Let furthermore Q_q be in the computation, with $q \geq i$. Then, necessarily,

   - $|Q_{i-1}|^q_\pi = (T_{i-1}, S_{i-1})\phi^q_{i-1}$ with
     $S_{i-1} = [\![h_1 \sharp i_1, \ldots, h_n \sharp i_n]\!] \uplus S$ and
     $T_{i-1} = [\![(R, h_1 \sharp i_1, \ldots, h_n \sharp i_n)]\!] \uplus T$,

- $|Q_i|_\pi^q = (T_i, S_i)\sigma_i\theta_i\phi_i^q$ with
  $S_i = [\![D_1\sharp\nu, \ldots, D_u\sharp(\nu+u-1)]\!] \uplus S_{i-1}$ and
  $T_i = T \uplus T^A$, with $T^A = T^A_{([\![D_1\sharp\nu,\ldots,D_u\sharp(\nu+u-1)]\!],S_{i-1})}$, and
- $\phi_{i-1}^q = \sigma_i\theta_i\phi_i^q$.

It can easily be verified that the second component of the CHR state representation for propagation remains finite.

For the first component we have the following. Let $\chi_{i-1}^q = \sigma_i\theta_i\chi_i^q$ be the substitution obtained by restricting the domain of $\phi_{i-1}^q = \sigma_i\theta_i\phi_i^q$ to the variables of R. Considering the precondition of the RC for CHR on propagation rules (see Corollary 3.2.1), since both the call set and the extended success set are closed under substitution, we have for $q \geq i$:

$$\forall H \in [\![H_1, \ldots, H_n]\!]\chi_{i-1}^q : H \in Call(P, I), \text{ and}$$
$$\forall G \in [\![G_1, \ldots, G_k]\!]\chi_{i-1}^q : G \in R^{SSE}_{rel(G)}.$$

As furthermore, by fairness, there must exists a state Q_{q_f} in the considered computation, with $q_f \geq i$, such that the body built-in constraints $[\![B_1, \ldots, B_l]\!]$ in $[\![D_1, \ldots, D_u]\!]$ of R were all solved successfully, we have that for any $q' \geq q_f$:

$$\forall B \in [\![B_1, \ldots, B_l]\!]\chi_{i-1}^{q'} : B \in R^{SSE}_{rel(B)}.$$

Therefore, a $q_f \geq i$ exists, such that for any $q' \geq q_f$, $\chi_{i-1}^{q'}$ satisfies the precondition of the RC for CHR on propagation rules. Under satisfaction of the RC for CHR on propagation rules, this implies that a $q_f \geq i$ exists, such that for any $q' \geq q_f$: for all $D \in [\![D_1, \ldots, D_u]\!]\chi_{i-1}^{q'}$ and all $H \in [\![H_1, \ldots, H_n]\!]\chi_{i-1}^{q'}$ holds that $H \succ_P D$, where $(\succeq_P, \succ_P)$ is the natural extension of $(\succeq, \succ)$ (see Corollary 3.2.1).

Thus, $\forall t \in T^A\chi_{i-1}^{q'} : (R, h_1\sharp i_1, \ldots, h_n\sharp i_n)\chi_{i-1}^{q'} \succ_\tau t$, because all tokens of T^A contain a reference to an added constraint of R. Therefore, $\forall t \in T^A\phi_{i-1}^{q'} : (R, h_1\sharp i_1, \ldots, h_n\sharp i_n)\phi_{i-1}^{q'} \succ_\tau t$. By multiset extensions, $[\![(R, h_1\sharp i_1, \ldots, h_n\sharp i_n)]\!]\phi_{i-1}^{q'} \succ_{\mu_\tau} T^A\phi_{i-1}^{q'}$, and by constructiveness, $[\![(R, h_1\sharp i_1, \ldots, h_n\sharp i_n)]\!]\phi_{i-1}^{q'} \uplus T\phi_{i-1}^{q'} \succ_{\mu_\tau} T\phi_{i-1}^{q'} \uplus T^A\phi_{i-1}^{q'}$. So, the first component of the CHR state representation for propagation decreases in size: $T_{i-1}\phi_{i-1}^{q'} \succ_{\mu_\tau} T_i\phi_i^{q'}$. Therefore, $|Q_{i-1}|_\pi^{q'} \succ_\pi |Q_i|_\pi^{q'}$.

By the same argument as used at the end of part 1, there also exists a q, such that for all $q' \geq q$: $|Q_{i-1}|_\pi^{q'} \approx_\pi |Q_{i-1}|_\pi^q \succ_\pi |Q_i|_\pi^q \approx_\pi |Q_i|_\pi^{q'}$.

□

We are ready to express the correctness of the RC for CHR on propagation rules, i.e., the RC for CHR on propagation rules guarantees propagation safeness of the CHR program P for the query set I.

Theorem 4.2.2. *If a CHR program P for a query set I satisfies the RC for CHR on propagation rules, then P is propagation safe for I.* □

Proof. Restricting attention to propagation sequences in CHR computations, the proof is a simple adaption of the proof of Theorem 2.3.1 using Lemma 4.2.2 instead. □

4.3 Termination of CHR

From now on, we assume propagation safeness of a CHR program P for a query set I. Thus, all full propagation sequences in computations of P for I are finite.

4.3.1 A new CHR state representation

To describe the effect of full propagation, we consider the constraints added by full propagation on the current state and collect these in a *propagation store*.

For the Primes example of Section 4.1, as can be verified, full propagation is well-defined. For any given state, the effect of full propagation is unique. In general, however, full propagation on a CHR state is not well-defined. For one, the guard of a propagation rule might be satisfiable for different answer substitutions. Secondly, depending on the order of application, the same constraints can be added with different labels. Thirdly, by solving built-in constraints (and thus also those in guards of propagation rules), bindings can be generated that enable or disable the applicability of propagation rules.

Therefore, to characterise full propagation, we require an exhaustive description of the effect of full propagation. That is, we collect the labelled built-in and CHR constraints that are added to the constraint store in a multiset, considering all full propagation sequences of P for some CHR state Q.

Definition 4.3.1 (propagation store)**.** *Let P be a propagation safe CHR program and I a query set. Let Q_i be any CHR state in a computation of P for I. Let $F_j = Q_i \xrightarrow{P} Q_{i+1} \xrightarrow{P} \ldots \xrightarrow{P} Q_q$ be any full propagation sequence of P for Q_i. Furthermore, let U_i^j be the set of labelled built-in and CHR constraints that are added to the constraint store by transitions in F_j. Then, the* propagation

store U_i *of* P *for* Q_i *is the multiset obtained as the join of all (multi)sets* U_i^j, *one for each full propagation sequence* F_j *of* P *for* Q_i. □

We need to guarantee that the propagation store is finite, such that we can use the propagation store to prove multiset decreases on (see Proposition 2.3.1). Therefore, we need the following property of a propagation safe CHR program.

Proposition 4.3.1 (finiteness of full propagation)**.** *Let* P *be a propagation safe CHR program and* I *a query set. Let* Q_i *be any CHR state in a computation of* P *for* I. *Then, the set* $\{F \mid F \text{ is a full propagation sequence of } P \text{ for } Q_i\}$ *is finite.* □

Proof. For any finite state Q_i in a computation of a propagation safe CHR program P, there are only a finite number of next states given by $\rightarrow_P$. That is, any call to the host language is by assumption universally terminating and there are only a finite number of combinations of constraints in the constraint store that can fire a CHR rule or can be solved by the host language.

Consider now a (computation) tree, with root node Q_i and child nodes the alternatives for next states as produced by propagation or solve transitions. Then, each branch in the tree corresponds to a full propagation sequence of P for Q_i. Since, by propagation safeness, the tree is depth-bound and, as shown above, there cannot be infinite branching, the tree remains finite. Thus, there can only be a finite number of alternative full propagation sequences of P for Q_i, each corresponding to a branch of the (computation) tree. □

Thus, for a propagation safe CHR program, the propagation store is finite.

Proposition 4.3.2 (finiteness of the propagation store)**.** *Let* P *be a propagation safe CHR program and* I *a query set. Let* Q_i *be any CHR state in a computation of* P *for* I. *Let* U_i *be the propagation store of* Q_i *w.r.t.* P. *Then,* U_i *is finite.* □

Proof. By Proposition 4.3.1, there can only be a finite number of full propagation sequences for any CHR state in any computation of a propagation safe CHR program P. Since in each of these sequences only a finite number of constraints can be added, we obtain the result. □

We now come to the heart of our approach. As we have already illustrated in Section 4.1, the constraint store combined with the propagation store remains constant for propagation transitions, while the size of the propagation forecast decreases. We furthermore have illustrated that the combined size of constraint

and propagation stores can be shown to decrease for simpagation, while the propagation forecast remains finite. We can therefore define a lexicographical ordering on a new representation of CHR states, that can be seen to decrease for every application of a CHR rule.

Definition 4.3.2 (CHR state representation)**.** *Let P be a propagation safe CHR program and I a query set. Let $Q_0 \xrightarrow{P,\phi_1} Q_1 \xrightarrow{P,\phi_2} Q_2 \xrightarrow{P,\phi_3} \dots \xrightarrow{P,\phi_q} Q_q \xrightarrow{P,\phi_{q+1}} \dots$ be any computation of P for I, where $\phi_i = \theta_i$ —the answer substitution for the built-in constraint— if $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ is a solve transition, and $\phi_i = \sigma_i\theta_i$ —the composition of the match substitution and the answer substitution for the guard— if $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ is a propagation or a simpagation transition. Then, for $q \geq i$, we define*

- $\phi_i^q = \phi^\emptyset \phi_{i+1}\phi_{i+2}\dots\phi_q$, *with $\phi^\emptyset$ the empty substitution, and*
- $|Q_i|_\omega^q = |\langle S_i, T_i\rangle_{\nu_i}|_\omega^q = (S_i\phi_i^q \uplus U_i\phi_i^q, T_i\phi_i^q)$. □

Note that the CHR state representation from above differs from the CHR state representation for termination of propagation (see Definition 4.2.5). That is, the CHR state representation from above is useful to verify decreases on consecutive program states for program termination, while the CHR state representation from Definition 4.2.5 is useful to prove propagation safeness.

We define the following ordering on the CHR state representation.

Definition 4.3.3 (CHR state ordering)**.** *Let $(\succeq_{\mu_P}^{\mathbb{N}}, \succ_{\mu_P}^{\mathbb{N}})$ be the multiset extension of a reduction pair $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ on $Atom_P^{\mathbb{N}}$ and $(\succeq_{\mu_\tau}, \succ_{\mu_\tau})$ the multiset extension of a reduction pair $(\succeq_\tau, \succ_\tau)$ on $Token_P$, based on $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$. Then, $\succeq_\omega$ is the lexicographical ordering on ordered pairs $(S \uplus U, T)$ of the CHR state representation, where the first element is compared on the basis of $(\succeq_{\mu_P}^{\mathbb{N}}, \succ_{\mu_P}^{\mathbb{N}})$ and the second on the basis of $(\succeq_{\mu_\tau}, \succ_{\mu_\tau})$.* □

The CHR state ordering yields a reduction pair $(\succeq_\omega, \succ_\omega)$.

Proposition 4.3.3. *Let $(\succeq_{\mu_P}^{\mathbb{N}}, \succ_{\mu_P}^{\mathbb{N}})$ be the multiset extension of a reduction pair $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ on $Atom_P^{\mathbb{N}}$ and $(\succeq_{\mu_\tau}, \succ_{\mu_\tau})$ the multiset extension of a reduction pair $(\succeq_\tau, \succ_\tau)$ on $Token_P$, based on $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$. Finally, let $\succeq_\omega$ be the CHR state ordering based on these extensions. Then, $(\succeq_\omega, \succ_\omega)$ is a reduction pair, where $\succ_\omega$ is the strict partial order associated to $\succeq_\omega$.* □

Proof. It is well-known that a lexicographical ordering, based on reduction pairs, yields a reduction pair (see e.g., [DM79] and [Der82]). □

Next, we introduce sufficient conditions for CHR rules of any kind. These conditions imply decreases on the CHR state representation of consecutive CHR states, in computations of a propagation safe CHR program P for a query set I.

4.3.2 The RC for CHR

As the RC for CHR on propagation rules (see Definition 4.2.3) is also sufficient to prove program termination, now, we want to introduce a condition on simpagation rules such that for every application of a simpagation rule the combined size of constraint and propagation store decreases. However, we do not want to reason explicitly on the propagation store. Full propagation may require many transitions and we do not wish to compute them all.

Assuming satisfaction of the RC for CHR on propagation, it turns out that it is sufficient to only consider the *first-layer propagation* of full propagation. Informally, these are the constraints of the propagation store that are added by propagation transitions using tokens that are present in the current propagation forecast (see Definition 4.3.9).

If the constraints added as a consequence of the tokens present in the current propagation forecast cannot undo a multiset decrease, neither will any of the constraints added as a consequence of newly introduced tokens, since these tokens (of further propagation layers), under satisfaction of the RC for CHR on propagation rules, correspond to the addition of even smaller constraints.

Our RC is a refinement of the RC for simplification rules from Definition 2.3.16. That is, we additionally impose that whenever a multiset decrease exists between the head and the body of a simpagation rule, all first-layer propagation that can follow the application of a simpagation rule and which uses added constraints of the simpagation rule —corresponding to the newly introduced tokens— cannot undo the multiset decrease.

To characterise this, we need the *decreasing ranks set* of a strict multiset decrease. To be able to define the notion of a decreasing ranks set of a strict multiset decrease, we first introduce *(maximal) decreasing equivalence multisubsets* of a strict multiset decrease.

Definition 4.3.4 (decreasing equivalence multisubset)**.** *Let U be a universe for the multisets M_A and M_B. Let $(\succeq, \succ)$ be a reduction pair on U and $(\succeq_\mu, \succ_\mu)$ the multiset extension of $(\succeq, \succ)$. Let $M_A \succ_\mu M_B$. A $(\succeq, \succ)$-equivalence multisubset $E = E_A \uplus E_B$ of $M_A \uplus M_B$, with $E_A \sqsubseteq M_A$ and $E_B \sqsubseteq M_B$, is* decreasing *iff $\sharp E_A > \sharp E_B$.* □

Definition 4.3.5 (maximal decreasing equivalence multisubset)**.** *Let U be a universe for the multisets M_A and M_B. Let $(\succeq, \succ)$ be a reduction pair on U and $(\succeq_\mu, \succ_\mu)$ the multiset extension of $(\succeq, \succ)$. Let $M_A \succ_\mu M_B$. A decreasing $(\succeq, \succ)$-equivalence multisubset $E = E_A \uplus E_B$ of $M_A \uplus M_B$, with $E_A \sqsubseteq M_A$ and $E_B \sqsubseteq M_B$, is* maximal *iff there does not exist a decreasing $(\succeq, \succ)$-equivalence multisubset $E' = E'_A \uplus E'_B$ of $M_A \uplus M_B$, with $E'_A \sqsubseteq M_A$ and $E'_B \sqsubseteq M_B$, such that there exists an $e \in E$ and an $e' \in E'$: $e' \succ e$.* □

We are now ready to define the notion of a decreasing ranks set of a strict multiset decrease.

Definition 4.3.6 (decreasing ranks set)**.** *Let U be a universe for the multisets M_A and M_B. Let $(\succeq, \succ)$ be a reduction pair on U and $(\succeq_\mu, \succ_\mu)$ the multiset extension of $(\succeq, \succ)$. Let $M_A \succ_\mu M_B$. A* decreasing ranks set *of $M_A \uplus M_B$ is a (multi)set $\mathcal{R} = [\![\rho_1, \ldots, \rho_r]\!]$, where $\mathcal{R} \sqsubseteq M_A$, containing one element ρ_i of each maximal decreasing equivalence multisubset E_i of $M_A \uplus M_B$.* □

We provide an example.

Example 4.3.1 (decreasing ranks set)**.** *Consider a reduction pair $(\succeq, \succ)$ for a universe $U = \{a, b, c, d, e\}$, where*

$$\succeq = \{(a,a), (b,b), (c,c), (d,d), (e,e), (a,c), (b,d), (d,b)\}$$

is a pre-order relation with associated strict partial order relation $\succ = \{(a,c)\}$. Let $(\succeq_\mu, \succ_\mu)$ be its multiset extension. Furthermore, consider two multisets $M_A = [\![a, a, b, c, d, e, e]\!]$ and $M_B = [\![a, c, c, b, e, e]\!]$ of U. It can easily be verified that $M_A \succ_\mu M_B$. Then, $\mathcal{R} = [\![a, b]\!]$ is a decreasing ranks set of $M_A \succ_\mu M_B$ and $\mathcal{R}' = [\![a, d]\!]$ is another. □

Note that the *decreasing ranks* in a decreasing ranks set $\mathcal{R}$ of a strict multiset decrease $M_A \succ_\mu M_B$ can be regarded as the reason for the multiset decrease. They define the maximal equivalence classes for which a decrease in cardinality exists between M_A and M_B. Therefore, adding any finite number of elements that are strictly smaller than some decreasing rank of $\mathcal{R}$ to M_B will not undo the multiset decrease. We have the following property of multiset extensions.

Proposition 4.3.4. *Let U be a universe for the multisets M_A, M_B and S. Let $(\succeq, \succ)$ be a reduction pair on U such that $M_A \succ_\mu M_B$, where $(\succeq_\mu, \succ_\mu)$ is the multiset extension of $(\succeq, \succ)$. Furthermore, let $\mathcal{R} = [\![\rho_1, \ldots, \rho_r]\!]$ be a decreasing ranks set of $M_A \succ_\mu M_B$ and let S be such that for all $s \in S$ holds that there exists a $\rho \in \mathcal{R}$ for which $\rho \succ s$. Then, $M_A \succ_\mu M_B \uplus S$.* □

Proof. First note that if there exists a $\rho \in \mathcal{R}$ such that $\rho \succ s$, there cannot exist another $\rho' \in \mathcal{R}$ such that $s \succeq \rho'$ as this would either conflict with the guarantee that none of the decreasing ranks can be compared to each other (see Definition 4.3.6) or with the guarantee that $\succeq$ is transitive. Thus, for each element $s \in S$, even if the cardinality of its $(\succeq_\mu, \succ_\mu)$-equivalent multisubsets increases between M_A and M_B, there will always exist a $(\succeq_\mu, \succ_\mu)$-equivalent multisubset —related to one of the decreasing ranks— of larger elements, for which the cardinality decreases between M_A and M_B (see Definition 2.3.7). □

We furthermore have the following property of multiset extensions.

Proposition 4.3.5. *Let U be a universe for the multisets M_A, M_B and S. Let $(\succeq, \succ)$ be a reduction pair on U such that $M_A \succ_\mu M_B$, where $(\succeq_\mu, \succ_\mu)$ is the multiset extension of $(\succeq, \succ)$. Then, by constructiveness, $M_A \uplus S \succ_\mu M_B \uplus S$. Furthermore, let $\mathcal{R} = [\![\rho_1, \ldots, \rho_r]\!]$ be a decreasing ranks set of $M_A \succ_\mu M_B$. Then, $\mathcal{R}$ is a decreasing ranks set of $M_A \uplus S \succ_\mu M_B \uplus S$.* □

Proof. Considering the proof of Proposition 2.3.2, this is obvious. □

We are finally ready to give the RC for CHR on simpagation rules.

Definition 4.3.7 (RC for CHR on simpagation rules)**.** *Let P be a CHR program and I a query set. Let $(\succeq, \succ)$ be a reduction pair on $Call(P, I)$ such that $Call(P, I)$ is rigid w.r.t. $(\succeq, \succ)$. Furthermore, let $(\succeq_\mu, \succ_\mu)$ be the multiset extension of $(\succeq, \succ)$. Then, a CHR program P for I satisfies the* RC for CHR on simpagation rules *iff:*

- *For any substitution χ of a simpagation rule*

  $$R_s \ @\ H_1, \ldots, H_j \setminus H_{j+1}, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

  in P, such that

  $$\forall H \in [\![H_1, \ldots, H_n]\!]\chi : H \in Call(P, I),$$
  $$\forall G \in [\![G_1, \ldots, G_k]\!]\chi : G \in R^{SSE}_{rel(G)}, \text{ and}$$
  $$\forall B \in [\![B_1, \ldots, B_l]\!]\chi : B \in R^{SSE}_{rel(B)},$$

  holds that

  $$[\![H_{j+1}, \ldots, H_n]\!]\chi \succ_\mu [\![C \mid C \in [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi \wedge C \in Call(P, I)]\!].$$

- *Furthermore, let $\mathcal{R} = [\![\rho_1, \ldots, \rho_r]\!]$ be a decreasing ranks set of*

  $$[\![H_{j+1}, \ldots, H_n]\!]\chi \succ_\mu [\![C \mid C \in [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi \wedge C \in Call(P, I)]\!].$$

Then, for any substitution χ' of a propagation rule

$$R_p \ @ \ H'_1, \ldots, H'_{n'} \Rightarrow G'_1, \ldots, G'_{k'} \mid B'_1, \ldots, B'_{l'}, C'_1, \ldots, C'_{m'}.$$

in P, such that

$$\forall H' \in [\![H'_1, \ldots, H'_{n'}]\!]\chi' : H' \in Call(P, I),$$
$$\forall G' \in [\![G'_1, \ldots, G'_{k'}]\!]\chi' : G' \in R^{SSE}_{rel(G')}, \text{ and}$$
$$\forall B' \in [\![B'_1, \ldots, B'_{l'}]\!]\chi' : B' \in R^{SSE}_{rel(B')},$$

and such that there exists a constraint D:

$$D \in [\![C_1, \ldots, C_m]\!]\chi \text{ and } D \in [\![H'_1, \ldots, H'_{n'}]\!]\chi';$$

holds that $\forall C' \in [\![C \mid C \in [\![B'_1, \ldots, B'_{l'}, C'_1, \ldots, C'_{m'}]\!]\chi' \wedge C \in Call(P, I)]\!]$, $\exists \rho \in \mathcal{R} : \rho \succ C'$. □

What is expressed in the second condition is that each time that a head constraint in a propagation rule matches an added constraint of the simpagation rule, by addition of its body constraints, it cannot undo the multiset decrease of the simpagation rule. Since we are verifying a local condition, we do not have information on the contents of the constraint store, therefore, an added CHR constraint can introduce an arbitrary finite number of tokens and thus an arbitrary finite number of copies of the constraints in the body of propagation rules. Hence, we require that the added constraints of propagation rules, that can fire as a consequence of an added body CHR constraints of the simpagation rule, demonstrate a strict decrease w.r.t. some decreasing rank of the multiset decrease of the simpagation rule.

By natural extensions, we have the following.

Corollary 4.3.1 (RC for CHR on simpagation rules)**.** *Let P be a CHR program and I a query set. Let $(\succeq, \succ)$ be a reduction pair on $Call(P, I)$, such that $Call(P, I)$ is rigid w.r.t. $(\succeq, \succ)$. Furthermore, let $(\succeq_P, \succ_P)$ be the natural extension of $(\succeq, \succ)$ to $Atom_P$ and $(\succeq_{\mu_P}, \succ_{\mu_P})$ its multiset extension. Then, a CHR program P for I satisfies the* RC for CHR on simpagation rules *iff:*

- *For any substitution χ of a simpagation rule*

  $$R_s \ @ \ H_1, \ldots, H_j \setminus H_{j+1}, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

  in P, such that

  $$\forall H \in [\![H_1, \ldots, H_n]\!]\chi : H \in Call(P, I),$$

$$\forall G \in [\![G_1, \ldots, G_k]\!]\chi : G \in R^{SSE}_{rel(G)}, \text{ and}$$
$$\forall B \in [\![B_1, \ldots, B_l]\!]\chi : B \in R^{SSE}_{rel(B)},$$

holds that $[\![H_{j+1}, \ldots, H_n]\!]\chi \succ_{\mu_P} [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi$.

- *Furthermore, let* $\mathcal{R} = [\![\rho_1, \ldots, \rho_r]\!]$ *be a decreasing ranks set of*

$$[\![H_{j+1}, \ldots, H_n]\!]\chi \succ_{\mu_P} [\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!]\chi.$$

Then, for any substitution χ' *of a propagation rule*

$$R_p \ @ \ H'_1, \ldots, H'_{n'} \Rightarrow G'_1, \ldots, G'_{k'} \mid B'_1, \ldots, B'_{l'}, C'_1, \ldots, C'_{m'}.$$

in P*, such that*

$$\forall H' \in [\![H'_1, \ldots, H'_{n'}]\!]\chi' : H' \in Call(P, I),$$
$$\forall G' \in [\![G'_1, \ldots, G'_{k'}]\!]\chi' : G' \in R^{SSE}_{rel(G')}, \text{ and}$$
$$\forall B' \in [\![B'_1, \ldots, B'_{l'}]\!]\chi' : B' \in R^{SSE}_{rel(B')},$$

and such that there exists a constraint D*:*

$$D \in [\![C_1, \ldots, C_m]\!]\chi \text{ and } D \in [\![H'_1, \ldots, H'_{n'}]\!]\chi';$$

holds that $\forall C' \in [\![B'_1, \ldots, B'_{l'}, C'_1, \ldots, C'_{m'}]\!]\chi'$, $\exists \rho \in \mathcal{R} : \rho \succ_P C'$. □

Proof. The proof is a simple adaption of the proof of Corollary 2.3.1, applied on the RC for CHR on simpagation rules instead (see Definition 4.3.1). □

Finally, by combining both RCs for CHR w.r.t. the same reduction pair, we obtain a RC for CHR programs, sufficient to prove program termination.

Definition 4.3.8 (RC for CHR)**.** *Let* P *be a CHR program and* I *a query set. Let* $(\succeq, \succ)$ *be a reduction pair on* $Call(P, I)$ *such that* $Call(P, I)$ *is rigid w.r.t.* $(\succeq, \succ)$*. Then, a CHR program* P *for* I *satisfies the* RC for CHR *iff* P *for* I *satisfies the RC for CHR on propagation rules w.r.t.* $(\succeq, \succ)$ *and* P *for* I *satisfies the RC for CHR on simpagation rules w.r.t.* $(\succeq, \succ)$*.* □

Before we prove correctness of the RC for CHR, we discuss termination of the CHR program of Sections 3.3.4 and 4.1, outside the scope of the approaches of Chapters 2 and 3. Here, we only verify the RC for CHR at an intuitive level. In Section 4.4.4, we provide a more formal verification of the RC for CHR.

Example 4.3.2 (problem class 2 from Section 3.3.4)**.** *The following CHR program* P *is terminating for any initial CHR state.*

$$R_1 \ @ \ a(s(N)), a(N), a(N) \Leftrightarrow a(s(N)), a(N).$$
$$R_2 \ @ \ a(s(N)) \Leftrightarrow a(N).$$
$$R_3 \ @ \ a(s(s(N))), a(s(s(N))) \Rightarrow a(N).$$

Comparing constraints of the $a/1$ predicate based on their argument's term-size (see Definition 2.3.12), we can construct a reduction pair $(\succeq, \succ)$ on $Call(P, I)$ for which $Call(P, I)$ is rigid.

For R_1, it can be verified that there is a strict multiset decrease between the head and the body of the rule. Considering application of R_3 on the added CHR constraints of R_1, it can be verified that this strict multiset decrease cannot be undone. For R_2, it can be verified that there is a strict multiset decrease between the head and the body of the rule. Considering application of R_3 on the added CHR constraints of R_2, it can be verified that this strict multiset decrease cannot be undone. Finally, for R_3, it can easily be verified that the added CHR constraints of the rule are strictly smaller than the head constraints of the rule. Therefore, P is terminating for I. □

4.3.3 Correctness of the RC for CHR

Let $Q_{i-1} \rightarrow_P Q_i$ be any transition in a computation of a CHR program P for a query set I, where U_{i-1} is the propagation store of Q_{i-1} and U_i the propagation store of Q_i. Then, $U_{i-1} = U \uplus U^R$ and $U_i = U \uplus U^A$. Here, U^R represents the constraints removed from the propagation store U_{i-1} and U^A the constraints added to the propagation store U_{i-1} as a result of the transition.

Note that we will not explicitly define U, U^R and U^A, since it will not be needed in the proofs that follow. Moreover, U, U^R and U^A can always be determined by computing full propagation on Q_{i-1} and Q_i. We will however provide the following intuitions regarding U^R and U^A. That is, in a simpagation transition the constraints removed from the propagation store are related to the constraints removed from the constraint store. In a solve or propagation transition the constraints removed from the propagation store are related to the alternatives for propagation that are no longer valid. In a simpagation transition the constraints added to the propagation store are related to the constraints added to the constraint store. Finally, in a solve or propagation transition there are no constraints added to the propagation store as solve and propagation transitions are both considered in the effect of full propagation.

We will also need the notion of *first-layer propagation*.

Definition 4.3.9 (first-layer propagation)**.** *Let P be a CHR program and I a query set. For any CHR state Q_i in any computation $Q_0 \xrightarrow{P} Q_1 \xrightarrow{P} Q_2 \xrightarrow{P} \ldots$*

of P for I, let U_i be the propagation store of P for Q_i. Then, the first-layer propagation of U_i, denoted U_i^1, is the multisubset of constraints of U_i added to the constraint store by propagation transitions that use tokens of the current propagation forecast T_i of Q_i. We also define the complement of U_i^1, i.e., $U_i^{1'}$, as $U_i^1 \uplus U_i^{1'} = U_i$. □

Thus, the constraints added in further propagation layers, i.e., $U_i^{1'}$, are added by propagation rules that fire on tokens that are added to the propagation forecast as the result of propagation on constraints added in the first-layer propagation U_i^1. Note that we will also consider the first-layer propagation of the constraints added to the propagation store and denote it is as U^{A_1}.

To prove the correctness of the RC for CHR (see Definition 4.3.8), that is, satisfaction of the RC for CHR implies program termination, we use the CHR state representation of Definition 4.3.2. We will need the following lemma.

Lemma 4.3.1 (boundedness of $|.|_\omega^q$ in q)**.** *Let P be a propagation safe CHR program and I a query set. Let $Q_0 \xrightarrow{P,\phi_1} Q_1 \xrightarrow{P,\phi_2} Q_2 \xrightarrow{P,\phi_3} \ldots \xrightarrow{P,\phi_q} Q_q \xrightarrow{P,\phi_{q+1}} \ldots$ be any computation of P for I. Let $(\succeq, \succ)$ be a reduction pair on $Call(P,I)$ for which $Call(P,I)$ is rigid. Let $(\succeq_{\mu_P}^{\mathbb{N}}, \succ_{\mu_P}^{\mathbb{N}})$ be the multiset extension of a reduction pair $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$ on $Atom_P^{\mathbb{N}}$, obtained as the supernatural extension of $(\succeq, \succ)$ and let $(\succeq_{\mu_\tau}, \succ_{\mu_\tau})$ be the multiset extension of a reduction pair $(\succeq_\tau, \succ_\tau)$ on $Token_P$, based on $(\succeq_P^{\mathbb{N}}, \succ_P^{\mathbb{N}})$. Finally, let $(\succeq_\omega, \succ_\omega)$ be the CHR state reduction pair based on these extensions. Then, for any Q_i in the computation, there exists a $q \geq i$, with Q_q in the computation, such that for all $q' \geq q$, with $Q_{q'}$ in the computation, holds that $|Q_i|_\omega^q \approx_\omega |Q_i|_\omega^{q'}$.* □

Proof. $Q_i = \langle S_i, T_i \rangle_{\nu_i}$ is a finite state. Therefore, by propagation safeness, the propagation store U_i is also finite. Considering this, given rigidity of the call set, the proof is a simple adaption of the proofs of Lemmas 2.3.2 and 4.2.1. □

Using the CHR state representation, under satisfaction of the RC for CHR, for any transition in a non-failed computation, there exists a strict decrease between the consecutive program states. We have the following lemma.

Lemma 4.3.2. *Let P be a CHR program which satisfies the RC for CHR for a query set I w.r.t. $(\succeq, \succ)$ on $Call(P,I)$. Let $(\succeq_{\mu_P}^{\mathbb{N}}, \succ_{\mu_P}^{\mathbb{N}})$ be the supernatural extension of $(\succeq, \succ)$ to $Atom_P^{\mathbb{N}}$ and $(\succeq_\tau, \succ_\tau)$ the reduction pair on $Token_P$, based on $(\succeq_{\mu_P}^{\mathbb{N}}, \succ_{\mu_P}^{\mathbb{N}})$. Furthermore, let $(\succeq_\omega, \succ_\omega)$ be the reduction pair on the CHR state representation based on these extensions. Then, for any transition $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ in a non-failed computation $Q_0 \xrightarrow{P,\phi_1} Q_1 \xrightarrow{P,\phi_2}$*

$Q_2 \xrightarrow{P,\phi_3} \ldots \xrightarrow{P,\phi_q} Q_q \xrightarrow{P,\phi_{q+1}} \ldots$ *of* P *for* I*, holds that* $\exists q \geq i$ *such that* $\forall q' \geq q : |Q_{i-1}|_\omega^{q'} \approx_\omega |Q_{i-1}|_\omega^q \succ_\omega |Q_i|_\omega^q \approx_\omega |Q_i|_\omega^{q'}$. □

Proof. By satisfaction of the RC for CHR on propagation rules, Theorem 4.2.2 guarantees that P is propagation safe for I. There are three kinds of transitions to consider:

1. Let $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ be a solve transition in a non-failed computation of P for I and let Q_q be in the computation, with $q \geq i$. Then, necessarily,

   - $|Q_{i-1}|_\omega^q = (\llbracket b \sharp i_b \rrbracket \uplus S \uplus U \uplus U^R, T)\phi_{i-1}^q$,
   - $|Q_i|_\omega^q = (S \uplus U \uplus U^A, T)\theta_i\phi_i^q$ with $U^A = \llbracket\rrbracket$, and
   - $\phi_{i-1}^q = \theta_i\phi_i^q$.

   As can be verified, the second component of the CHR state representation remains constant. For the first component, as $\llbracket b \sharp i_b \rrbracket \phi_{i-1}^q \succ_{\mu_P}^{\mathbb{N}} \llbracket\rrbracket$, by constructiveness, $\llbracket b \sharp i_b \rrbracket \phi_{i-1}^q \uplus S\phi_{i-1}^q \uplus U\phi_{i-1}^q \succ_{\mu_P}^{\mathbb{N}} S\phi_{i-1}^q \uplus U\phi_{i-1}^q$. Thus, the first component decreases in size: $(\llbracket b \sharp i_b \rrbracket \uplus S \uplus U \uplus U^R)\phi_{i-1}^q \succ_{\mu_P}^{\mathbb{N}} (S \uplus U \uplus U^A)\phi_{i-1}^q$. Therefore, $|Q_{i-1}|_\omega^q \succ_\omega |Q_i|_\omega^q$.

   Since this holds for any $q \geq i$, and by Lemma 4.3.1 there exists a q_{i-1}, such that for all $q' \geq q_{i-1}$, $|Q_{i-1}|_\omega^{q'} \approx_\omega |Q_{i-1}|_\omega^{q_{i-1}}$, and a q_i, such that for all $q' \geq q_i$, $|Q_i|_\omega^{q'} \approx_\omega |Q_i|_\omega^{q_i}$, we have that for $q = max\{q_{i-1}, q_i\}$, for all $q' \geq q : |Q_{i-1}|_\omega^q \approx_\omega |Q_{i-1}|_\omega^q \succ_\omega |Q_i|_\omega^q \approx_\omega |Q_i|_\omega^{q'}$.

2. Let $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ be a propagation transition in a non-failed computation of P for I. Let $(R \ @ \ H_1, \ldots, H_n \Rightarrow G_1, \ldots, G_k \mid D_1, \ldots, D_u.)$ be the propagation rule of P applied in the transition and this on the CHR constraints $h_1, \ldots, h_n$ of Q_{i-1} with match substitution σ_i, such that the guards $G_1, \ldots, G_k$ all succeed with answer substitution θ_i. Let furthermore Q_q be in the computation, with $q \geq i$. Then, necessarily,

   - $|Q_{i-1}|_\omega^q = (S_{i-1} \uplus U_{i-1}, T_{i-1})\phi_{i-1}^q$ with
     $S_{i-1} = \llbracket h_1 \sharp i_1, \ldots, h_n \sharp i_n \rrbracket \uplus S$,
     $U_{i-1} = U \uplus \llbracket D_1 \sharp \nu, \ldots, D_u \sharp (\nu + u - 1) \rrbracket \uplus U^R$ and
     $T_{i-1} = \llbracket (R, h_1 \sharp i_1, \ldots, h_n \sharp i_n) \rrbracket \uplus T$,
   - $|Q_i|_\omega^q = (S_i \uplus U_i, T_i)\sigma_i\theta_i\phi_i^q$ with
     $S_i = \llbracket D_1 \sharp \nu, \ldots, D_u \sharp (\nu + u - 1) \rrbracket \uplus S_{i-1}$,
     $U_i = U \uplus U^A$ with $U^A = \llbracket\rrbracket$ and
     $T_i = T \uplus T^A_{(\llbracket D_1 \sharp \nu, \ldots, D_u \sharp (\nu+u-1) \rrbracket, S_{i-1})}$, and
   - $\phi_{i-1}^q = \sigma_i\theta_i\phi_i^q$.

It can easily be verified that the size of the first component of the CHR state representation can only decrease in size: $(S_{i-1} \uplus U_{i-1})\phi^q_{i-1} \succeq^{\mathbb{N}}_{\mu_P} (S_i \uplus U_i)\sigma_i\theta_i\phi^q_i$ for $q \geq i$. In fact, the combined size of constraint and propagation stores will usually not be constant, except if the effect of full propagation is well-defined (unique), such as for the Primes example of Section 4.1. In that case $U^R = [\![]\!]$.

For the second component, we may reuse the proof in part 2 of Lemma 4.2.2 to prove that $T_{i-1}\phi^{q'}_{i-1} \succ_{\mu_\tau} T_i\phi^{q'}_i$. Therefore, $|Q_{i-1}|^{q'}_\omega \succ_\omega |Q_i|^{q'}_\omega$.

By the same argument as used at the end of part 1, there also exists a q, such that for all $q' \geq q : |Q_{i-1}|^{q'}_\omega \approx_\omega |Q_{i-1}|^q_\omega \succ_\omega |Q_i|^q_\omega \approx_\omega |Q_i|^{q'}_\omega$.

3. Let $Q_{i-1} \xrightarrow{P,\phi_i} Q_i$ be a simpagation transition in a non-failed computation of P for I. Let $(R \ @ \ H_1, \ldots, H_j \setminus H_{j+1}, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid D_1, \ldots, D_u.)$ be the simpagation rule of P applied in the transition and this on the CHR constraints $h_1, \ldots, h_n$ of Q_{i-1} with match substitution σ_i, such that the guards $G_1, \ldots, G_k$ all succeed with answer substitution θ_i. Let furthermore Q_q be in the computation, with $q \geq i$. Then, necessarily,

   - $|Q_{i-1}|^q_\omega = (S_{i-1} \uplus U_{i-1}, T_{i-1})\phi^q_{i-1}$ with
     $S_{i-1} = [\![h_1 \sharp i_1, \ldots, h_n \sharp i_n]\!] \uplus S$,
     $U_{i-1} = U \uplus U^R$ and
     $T_{i-1} = T$,
   - $|Q_i|^q_\omega = (S_i \uplus U_i, T_i)\sigma_i\theta_i\phi^q_i$ with
     $S_i = [\![h_1 \sharp i_1, \ldots, h_j \sharp i_j, D_1 \sharp \nu, \ldots, D_u \sharp (\nu + u - 1)]\!] \uplus S$,
     $U_i = U \uplus U^A$ and
     $T_i = T \uplus T^A_{([\![D_1 \sharp \nu, \ldots, D_u \sharp (\nu+u-1)]\!], [\![h_1 \sharp i_1, \ldots, h_j \sharp i_j]\!] \uplus S)}$, and
   - $\phi^q_{i-1} = \sigma_i\theta_i\phi^q_i$.

   As can be verified, the second component of the CHR state representation remains finite.

   For the first component, we have the following. Let $\chi^q_{i-1} = \sigma_i\theta_i\chi^q_i$ be the substitution obtained by restricting the domain of $\phi^q_{i-1} = \sigma_i\theta_i\phi^q_i$ to the variables of R. Then, considering the precondition of the RC for CHR on simpagation rules (see Corollary 4.3.1), since both the call set and the extended success set are closed under substitution, we have for $q \geq i$:

   $$\forall H \in [\![H_1, \ldots, H_n]\!]\chi^q_{i-1} : H \in Call(P, I), \text{ and}$$
   $$\forall G \in [\![G_1, \ldots, G_k]\!]\chi^q_{i-1} : G \in R^{SSE}_{rel(G)}.$$

As furthermore, by fairness, there must exists a state Q_{q_f} in the considered computation, with $q_f \geq i$, such that the body built-in constraints $B_1, \ldots, B_l$ in $[\![D_1, \ldots, D_u]\!]$ of R were all solved successfully, we have that for any $q' \geq q_f$:

$$\forall B \in [\![B_1, \ldots, B_l]\!]\chi_{i-1}^{q'} : B \in R_{rel(B)}^{SSE}.$$

Therefore, a $q_f \geq i$ exists, such that for any $q' \geq q_f$, $\chi_{i-1}^{q'}$ satisfies the precondition of the RC for CHR on simpagation rules.

Let $(\succeq_P, \succ_P)$ be the natural extension of $(\succeq, \succ)$ and let $(\succeq_{\mu_P}, \succ_{\mu_P})$ be the multiset extension of $(\succeq_P, \succ_P)$. Then, under satisfaction of the RC for CHR, this implies that a $q_f \geq i$ exists, such that for any $q' \geq q_f$: $[\![H_{j+1}, \ldots, H_n]\!]\chi_{i-1}^{q'} \succ_{\mu_P} [\![D_1, \ldots, D_u]\!]\chi_{i-1}^{q'}$, with a decreasing ranks set $\mathcal{R} = [\![\rho_1, \ldots, \rho_r]\!] \sqsubseteq [\![H_{j+1}, \ldots, H_n]\!]\chi_{i-1}^{q'}$ or else $[\![h_{j+1}, \ldots, h_n]\!]\phi_{i-1}^{q'} \succ_{\mu_P} [\![D_1, \ldots, D_u]\!]\phi_{i-1}^{q'}$, with $\mathcal{R} \sqsubseteq [\![h_{j+1}, \ldots, h_n]\!]\phi_{i-1}^{q'}$.

By supernatural extensions,

$$[\![h_{j+1}\sharp i_{j+1}, \ldots, h_n\sharp i_n]\!]\phi_{i-1}^{q'} \succ_{\mu_P}^{\mathbb{N}} [\![D_1\sharp\nu, \ldots, D_u\sharp(\nu+u-1)]\!]\phi_{i-1}^{q'},$$

with $\mathcal{R}' = [\![\rho_1\sharp i_{\rho_1}, \ldots, \rho_r\sharp i_{\rho_r}]\!] \sqsubseteq [\![h_{j+1}\sharp i_{j+1}, \ldots, h_n\sharp i_n]\!]\phi_{i-1}^{q'}$ as a decreasing ranks set. Then, by constructiveness and Proposition 4.3.5,

$$([\![h_1\sharp i_1, \ldots, h_j\sharp i_j]\!] \uplus [\![h_{j+1}\sharp i_{j+1}, \ldots, h_n\sharp i_n]\!] \uplus S \uplus U)\phi_{i-1}^{q'} \succ_{\mu_P}^{\mathbb{N}}$$

$$([\![h_1\sharp i_1, \ldots, h_j\sharp i_j]\!] \uplus [\![D_1\sharp\nu, \ldots, D_u\sharp(\nu+u-1)]\!] \uplus S \uplus U)\phi_{i-1}^{q'},$$

with a decreasing ranks set $\mathcal{R}'$.

By the RC for CHR on propagation rules, a sufficiently large $q_p \geq q_f$ exists such that for any $q' \geq q_p$ and for any constraint in $U^A\phi_{i-1}^{q'}$, there exists a constraint in $U^{A_1}\phi_{i-1}^{q'}$ that is larger or equal to it. Since the RC for CHR on simpagation guarantees that all constraints of $U^{A_1}\phi_{i-1}^{q'}$ are strictly smaller in size than some decreasing rank in $\mathcal{R}'$, related to $\mathcal{R}$ by supernatural extensions, we have that for all constraints of $U^A\phi_{i-1}^{q'}$ there must also exist a decreasing rank in $\mathcal{R}'$ that is strictly larger. Therefore, by Proposition 4.3.4, a $q_p \geq q_f$ exists such that for any $q' \geq q_p$:

$$([\![h_1\sharp i_1, \ldots, h_j\sharp i_j]\!] \uplus [\![h_{j+1}\sharp i_{j+1}, \ldots, h_n\sharp i_n]\!] \uplus S \uplus U)\phi_{i-1}^{q'} \succ_{\mu_P}^{\mathbb{N}}$$

$$([\![h_1\sharp i_1, \ldots, h_j\sharp i_j]\!] \uplus [\![D_1\sharp\nu, \ldots, D_u\sharp(\nu+u-1)]\!] \uplus S \uplus U \uplus U^A)\phi_{i-1}^{q'}.$$

Thus, a $q_p \geq q_f$ exists such that for any $q' \geq q_p$:

$$([\![h_1 \sharp i_1, \ldots, h_j \sharp i_j]\!] \uplus [\![h_{j+1} \sharp i_{j+1}, \ldots, h_n \sharp i_n]\!] \uplus S \uplus U \uplus U^R)\phi_{i-1}^{q'} \succ_{\mu_P}^{\mathbb{N}}$$

$$([\![h_1 \sharp i_1, \ldots, h_j \sharp i_j]\!] \uplus [\![D_1 \sharp \nu, \ldots, D_u \sharp (\nu + u - 1)]\!] \uplus S \uplus U \uplus U^A)\phi_{i-1}^{q'}.$$

So, the first component of the CHR state representation decreases in size. Therefore, $|Q_{i-1}|_\omega^{q'} \succ_\omega |Q_i|_\omega^{q'}$.

By the same argument as used at the end of part 1, there also exists a q, such that for all $q' \geq q$: $|Q_{i-1}|_\omega^{q'} \approx_\omega |Q_{i-1}|_\omega^{q} \succ_\omega |Q_i|_\omega^{q} \approx_\omega |Q_i|_\omega^{q'}$.

□

We are ready to express the correctness of our RC, i.e., the RC for CHR guarantees termination of the CHR program P for the query set I.

Theorem 4.3.1. *If a CHR program P for a query set I satisfies the RC for CHR, then P is terminating for I.* □

Proof. A simple adaption of the proof of Theorem 2.3.1 using Lemma 4.3.2. □

4.4 Termination of typical CHR programs

In this section, we revisit a number of typical CHR programs and prove termination with the RC for CHR.

4.4.1 Merge-sort

The CHR program P,

$$R_1 \ @ \ msort([]) \Leftrightarrow true.$$
$$R_2 \ @ \ msort([L|Ls]) \Leftrightarrow r(0, L), msort(Ls).$$
$$R_3 \ @ \ r(D, L1), r(D, L2) \Leftrightarrow less(L1, L2) \mid r(s(D), L1), a(L1, L2).$$
$$R_4 \ @ \ a(L1, L2) \setminus a(L1, L3) \Leftrightarrow less(L2, L3) \mid a(L2, L3).$$

is the Merge-sort program from the previous chapters (see Section 2.4.2 and Section 3.3.1). Since the program does not contain propagation, the RC for CHR reduces to the RC for abstract CHR. That is, the condition on propagation rules and the condition on first-layer propagation can be ignored.

Therefore, the proof for satisfaction of the RC for CHR for the Merge-sort program is identical to the proof for satisfaction of the RC for abstract CHR of Section 2.4.2. Thus, P is terminating for I.

4.4.2 Primes

The CHR program P,

$$R_1 \ @ \ primes(M) \setminus primes(N) \Leftrightarrow div(M,N) \mid true.$$
$$R_2 \ @ \ primes(N) \Rightarrow N > 2 \mid Np \ is \ N-1, primes(Np).$$

is the Primes program from the previous chapter (see Section 3.3.2). The intended use I, the call set $Call(P,I)$ and the extended success sets $R^{SSE}_{>/2}$ and $R^{SSE}_{is/2}$ are all given in Section 3.3.2. Therefore, we do not repeat them here. Furthermore, we reuse the reduction pair $(\succeq, \succ)$, as defined in Section 3.3.2, for which $Call(P,I)$ is rigid.

To prove satisfaction of the RC for CHR w.r.t. $(\succeq, \succ)$, for every application of R_1, we verify a strict multiset decrease between the head and the body of the rule. This is obviously the case as the body of R_1 is empty. Furthermore, since the body of R_1 is empty there cannot be first-layer propagation on the added constraints of R_1 and thus the condition on first-layer propagation is trivially satisfied. For R_2, we verify a strict decrease between all constraints used in the head and all constraints added in the body of the propagation rule. This is the case as for any χ, where $primes(N)\chi \in Call(P,I) \wedge \ (N > 2)\chi \in R^{SSE}_{rel(N>2)} \wedge (Np \ is \ N-1)\chi \in R^{SSE}_{rel(Np \ is \ N-1)}$, holds that $primes(N)\chi \succ primes(Np)\chi$. Thus, P is terminating for I.

4.4.3 Fibonacci

The CHR program P,

$$R_1 \ @ \ fib(N,M) \Rightarrow N = 0 \mid M = 0.$$
$$R_2 \ @ \ fib(N,M) \Rightarrow N = s(0) \mid M = 1.$$
$$R_3 \ @ \ fib(s(s(N)), M_1), fib(s(N), M_2) \Rightarrow fib(N,M), M_1 \ is \ M_2 + M.$$

is the Fibonacci program from the previous chapter (see Section 3.3.3). The intended use I, the call set $Call(P,I)$ and the extended success sets $R^{SSE}_{=/2}$ and $R^{SSE}_{is/2}$ are all given in Section 3.3.3. Therefore, we do not repeat them here.

Furthermore, we reuse the reduction pair $(\succeq, \succ)$, as defined in Section 3.3.3, for which $Call(P, I)$ is rigid.

Since the program does not contain simpagation, the RC for CHR reduces to the RC for CHR with propagation on propagation rules. Therefore, the proof for satisfaction of the RC for CHR for the Fibonacci program is identical to the proof for satisfaction of the RC for CHR with propagation of Section 3.3.3. Thus, P is terminating for I.

4.4.4 Second problem from Section 3.3.4

The CHR program P,

$$
\begin{aligned}
&R_1 \ @\ a(s(N)), a(N), a(N) \Leftrightarrow a(s(N)), a(N).\\
&R_2 \ @\ a(s(N)) \Leftrightarrow a(N).\\
&R_3 \ @\ a(s(s(N))), a(s(s(N))) \Rightarrow a(N).
\end{aligned}
$$

is the second problem of Section 3.3.4 that cannot be proven terminating using the approaches of previous chapters.

Applying the RC for CHR, we can prove termination. If the intended use I of P is characterised by $Atom_P$, we can derive a call set $Call(P, I) = I$. Comparing constraints of the $a/1$ predicate based on their argument's term-size (see Definition 2.3.12), we can construct a reduction pair $(\succeq, \succ)$ on $Call(P, I)$, such that $Call(P, I)$ is rigid w.r.t $(\succeq, \succ)$.

For R_1, for any χ, we can verify a strict multiset decrease with decreasing ranks set $\mathcal{R} = \{a(N)\chi\}$. Considering the constraints added in the body of R_1, i.e., $a(s(N))\chi$ and $a(N)\chi$, then by propagation on the added constraints, we can only introduce constraints that are strictly smaller than $a(N)\chi$ of $\mathcal{R}$. For R_2, for any χ, we can verify a strict multiset decrease with decreasing ranks set $\mathcal{R}' = \{a(s(N))\chi\}$. Considering the constraints added in the body of R_2, i.e., $a(N)\chi$, then by propagation on the added constraints we can only introduce constraints that are strictly smaller than $a(s(N))\chi$ of $\mathcal{R}'$. Finally, for R_3, for any χ, all head constraints are strictly greater than all body constraints. Thus, P is terminating for I.

Chapter 5

Theory: Conclusions

In the previous chapters, we worked towards a theoretical framework for termination analysis of the full CHR language, for which decreases can be verified on a new representation of CHR states. Since CHR is often built on top of a host language, we considered that any call to the host language of CHR is universally terminating and that such calls can only result in variable bindings. As such, we were able to focus on the termination behaviour of CHR.

In Chapter 2, we discussed a generalisation of the approach of [Frü00], successful on CHR programs without propagation. In contrast to the rather informal discussion of [Frü00], Chapter 2 provides a formal framework for verifying state-size decreases on the constraint store at the level of the transitions of a CHR program. Although the approach discussed in Chapter 4 is a direct generalisation of the approach of Chapter 2, Chapter 2 served the purpose of introducing several of the aspects of termination analysis of CHR in light of the next chapters.

In Chapter 3, the first approach to termination analysis of CHR programs with propagation is discussed. It is a novel approach, introducing a new characterisation of CHR-termination based on the finite addition of constraints. The approach of Chapter 3 generalises the approach of [VDSP08]. That is, we considered general orderings and a call set based on used constraints.

As explained in Section 3.3.4, the approaches of Chapters 2 and 3 are complementary. That is, the approach of Chapter 2 is not applicable to CHR programs with propagation. It is however more powerful than the approach of

Chapter 3 on CHR programs that do not contain propagation. Because of this, but also because there are interesting CHR programs that cannot be handled by either one of these approaches, a new more general approach was introduced in [PDS08a]. This new approach is the subject of Chapter 4.

The approach of Chapter 4 characterises the termination problem for CHR well. By considering the propagation store of a state, representing the effect of full propagation on a state, a new CHR state representation is obtained that can be seen to decrease for terminating CHR programs with propagation. That is, for propagation in terminating CHR programs, the combined size of constraint and propagation store cannot increase while the size of the propagation forecast can be seen to decrease. For simpagation in terminating CHR programs, the combined size of constraint and propagation store can be seen to decrease while the size of the propagation forecast remains finite.

The RC for CHR of Chapter 4 guarantees such decreases on this new state representation. The condition on propagation rules guarantees decreases on the propagation forecast, corresponding to the restriction that a propagation rule can never introduce CHR constraints that through further propagation can replace constraints used in the head of the propagation rule. In the presence of a fire-once policy, such a notion captures the termination problem of propagation in CHR well. On the other hand, the condition on simpagation rules of Chapter 4 guarantees decreases on the combined size of constraint and propagation store. Its meaning corresponds to the restriction that a multiset decrease must exist for any application of a simpagation rule such that the decrease can never be undone by propagation on the added constraints of the rule. This notion captures the termination problem of simpagation (w.r.t. propagation) well.

This chapter concludes the discussion on theoretical frameworks for termination analysis of CHR. First, we provide a comparison of the different approaches discussed in the previous chapters. Then, we discuss their limitations and provide intuitions to overcome these limitations. As such, we outline future work regarding improved theoretical frameworks.

5.1 A comparison of RCs

In this section, we compare the power of the different RCs, introduced in Chapters 2, 3 and 4, w.r.t. the theoretical semantics of CHR.

5.1.1 RC for abstract CHR vs. RC for CHR with propagation

As explained in Section 3.3.4, the approaches of Chapters 2 and 3 are complementary. Comparing the condition on simplification rules of Chapter 2 with the condition on simpagation rules of Chapter 3, we can verify that the condition on simpagation rules in the RC for CHR with propagation (see Definition 3.2.5) is an instance of the condition on simplification rules of the RC for abstract CHR (see Definition 2.3.16).

More precisely, it is that instance for which a strict multiset decrease exists on the simpagation rule, such that the constraints in the decreasing ranks set (see Definition 4.3.6) of the strict multiset decrease are all in the $(\succeq, \succ)$-maximal multisubsets (see Definition 3.2.4) of the removed and added constraints of the rule. Thus, we have the following property.

Proposition 5.1.1. *Let P be a CHR program without propagation and let I be a query set. If the RC for CHR with propagation can be satisfied for P with I using $(\succeq, \succ)$ on $Call(P, I)$, then the RC for abstract CHR can be satisfied for P with I using $(\succeq, \succ)$ on $Call(P, I)$.* □

Proof. First note that the abstract and theoretical CHR semantics are equivalent if there is no propagation. Then, consider, in the RC for abstract CHR (expressed on simplification rules only), that the kept heads of a simpagation rule are part of the decrease conditions, but that they will appear on both sides of these conditions. Hence, by constructiveness, we can ignore them. Then, the RC for CHR with propagation on simpagation rules can easily be verified to be an instance of the RC for abstract CHR. □

Furthermore, it is obvious that for a CHR program with propagation, only the RC for CHR with propagation is applicable. Therefore, given the first problem of Section 3.3.4, we can conclude that for the theoretical CHR semantics both approaches are complementary w.r.t. the presence of propagation rules.

5.1.2 RC for abstract CHR vs. RC for CHR

It should be clear that the RC for CHR of Chapter 4 is strictly more powerful than the RC for abstract CHR of Chapter 2 (see Section 3.3.4 and Section 4.3.2). When no propagation is present, the RC for CHR on simpagation rules (see Definition 4.3.7) reduces to the RC for abstract CHR (see Definition 2.3.16). Thus, we have the following property.

Proposition 5.1.2. *If the RC for abstract CHR can be satisfied for a CHR program P with query set I using $(\succeq, \succ)$ on $Call(P, I)$, then the RC for CHR can be satisfied for P with I using $(\succeq, \succ)$ on $Call(P, I)$.* □

Proof. First, note that the abstract and theoretical CHR semantics are equivalent if there is no propagation. Then, consider, in the RC for abstract CHR, that the kept heads of a simpagation rule are part of the decrease conditions, but that they will appear on both sides of these conditions, hence we can ignore them. Then, it is easily verified that without propagation the RC for CHR on simpagation rules reduces to a condition that is equivalent to the condition on simplification rules of the RC for abstract CHR. □

Furthermore, for a CHR program with propagation, only the RC for CHR is applicable. Therefore, we may conclude that for the theoretical CHR semantics the RC for CHR is strictly more powerful than the RC for abstract CHR.

5.1.3 RC for CHR with propagation vs. RC for CHR

When comparing the RC for CHR of Chapter 4 to the RC for CHR with propagation from Chapter 3, we first consider that the condition on propagation rules of the RC for CHR with propagation (see Definition 3.2.5) is identical to the RC for CHR on propagation rules (see Definition 4.2.3). Comparing the conditions on simpagation rules, we can verify that the condition on simpagation rules of the RC for CHR with propagation (see Definition 3.2.5) is an instance of the RC for CHR on simpagation rules (see Definition 4.3.7).

More precisely, it is that instance for which a strict multiset decrease exists on the simpagation rule, such that the constraints in the decreasing ranks set of the strict multiset decrease are all in the $(\succeq, \succ)$-maximal multisubsets of the removed and added constraints of the rule. In this particular instance of a multiset decrease, first-layer propagation on the added constraints of the CHR rule can never undo the multiset decrease. Hence, in the condition on simpagation rules of the RC for CHR with propagation, there is no condition on first-layer propagation required. Thus, we have the following property.

Proposition 5.1.3. *If the RC for CHR with propagation can be satisfied for a CHR program P with query set I using $(\succeq, \succ)$ on $Call(P, I)$, then the RC for CHR can be satisfied for P with I using $(\succeq, \succ)$ on $Call(P, I)$.* □

Proof. For both RCs, the condition on propagation rules are identical. For the condition on simpagation rules, consider first that the condition on simpagation

rules in the RC for CHR with propagation is an instance of the condition on simpagation rules in the RC for CHR. Then, consider that the condition on simpagation rules in the RC for CHR with propagation relates to that instance of a multiset decrease, where the decreasing ranks set is such that no constraint, added in the body of the simpagation rule, is strictly larger than some constraint in the decreasing ranks set. In this particular instance of a multiset decrease, under satisfaction of the RC for CHR on propagation rules, further propagation on the added constraints of a simpagation rule can never undo the multiset decrease. Therefore, for this instance, we may ignore first-layer propagation in the RC for CHR. □

Thus, considering Section 3.3.4 and Section 4.4.4, where we give an example of a program that cannot be handled by the RC for CHR with propagation but can be handled by the RC for CHR, we may conclude that the RC for CHR is strictly more powerful than the RC for CHR with propagation.

5.2 Limitations of the RC for CHR

The approach of Chapter 4 captures the termination problem for CHR well. There are however a number of limitations to our approach.

5.2.1 The success set of CHR predicates

One limitation to our approach is that we cannot handle CHR constraints as intermediate calls. Thus, if termination depends on CHR constraints used as intermediate calls, our approach cannot prove termination. Since such programs may occur in practice (see Example 2.3.13), it is therefore interesting to regard an approach that can consider intermediate CHR constraints.

In developing such an approach, a number of problems need to be addressed first. In contrast to LP with SLD resolution, for the theoretical CHR semantics there is no selection or application rule defined. Thus, in CHR, there is no way of determining intermediate calls based on a selection or application rule. Therefore, in order to be able to use the success set of added CHR constraints, we need, besides the guarantee of a fair semantics, a proof of termination of the predicate to which the CHR constraint belongs.

To do so, first of all, we will need a notion for a terminating CHR predicate. In CHR, it is not entirely clear what such a notion should mean, however, intuitively, it corresponds to the absence of infinite computations involving the

CHR predicate. Therefore, as a simple adaption of such a notion in LP, it may be defined as termination of the Strongly Connected Components (SCCs) of the CHR program in which the CHR predicate occurs (see Chapter 7).

Secondly, we will also need a modular approach to termination analysis of CHR, based on the SCCs of the program. This, as we will discuss in Chapter 7, can be done similarly to such an approach for LP [BCER02, NDSGSK11]. Using this modular approach, if after a first attempt not all SCCs can be proven terminating, we can derive the terminating CHR predicates related to the terminating SCCs. Then, in next attempts on the remaining SCCs, we can take the success set of terminating CHR predicates into account.

5.2.2 Extensions different from multiset extensions

A second, more fundamental, limitation to our approach is related to a requirement of extensions different from the multiset extensions on which our RCs are based (see Section 2.3.2).

An additive extension

Consider the CHR program

$$R_1 \ @ \ a, a \Leftrightarrow b. \qquad\qquad R_2 \ @ \ b \Leftrightarrow a.$$

Clearly it is terminating for any query to it. However, it cannot be proven terminating using the RC for CHR: for the first rule a needs to be greater than or equal to b and for the second rule b needs to be strictly greater than a.

In [Frü00], a second approach to termination analysis of abstract CHR is discussed, based on an *additive extension*. In an additive extension, state-size decreases are verified on the basis of adding the sizes of constraints of the state together. Such an extension is constructive and thus allows for local conditions. As can be verified, for the example above, if we measure a by the number 2 and b by 3 then a decrease can be shown for each rule application.

Alternatively, the transformational approach of [PSDS07a, PSDS07b] can also be used to handle the program above. Finally, the approach based on self-sustainability of SCCs in CHR can also prove termination (see Chapter 7).

A multiplicative extension

A second kind of CHR programs, that cannot be proven terminating with the RC for CHR, can be characterised by the CHR rule

$$l([A|LA]), l([B|LB]) \Leftrightarrow \\ len(LA, LenA), len(LB, LenB), LenA \geq LenB \mid l([B, A|LA]), l(LB).$$

The rule concatenates lists. It collects the elements of several lists into a single list and does so by adding to a list of elements, elements of shorter lists.

We cannot prove termination of this program using the RC for CHR. That is, the length of the longest list becomes longer and the length of the smallest list shorter. Furthermore, it cannot be proven terminating using the additive extension of the previous subsection. However, if we introduce a multiplicative extension, similar to the additive extension introduced above, it can be proven terminating. That is, if we measure the size of the constraints by list-length, then for each consecutive state we can show a decrease on the multiplicative extension: $LenA \cdot LenB > (LenA + 1) \cdot (LenB - 1)$ because $LenA \geq LenB \geq 1$.

A transformation of single-headed propagation rules

Finally, consider the CHR program

$$R_1 \ @ \ a, a, a \Leftrightarrow b. \qquad\qquad R_2 \ @ \ b \Rightarrow a.$$

The RC for CHR cannot prove termination of it. Furthermore, since there is propagation, we can no longer use the additive or multiplicative extension from above. Furthermore, the transformational approach of [PSDS07a, PSDS07b] and the approach of Chapter 7 will not be applicable either.

One way to tackle this problem is to replace single-headed propagation rules by single-headed simplification rules as in [PSDS07a, PSDS07b]. This is done by encoding a fire-once policy for single-headed propagation rules by means of flags, local to the constraints used in single-headed propagation rules. We illustrate this next, on the example above.

The program above is equivalent to the program

$$R_1 \ @ \ a, a, a \Leftrightarrow b(1). \qquad\qquad R_2 \ @ \ b(1) \Leftrightarrow b(0), a.$$

Here, $b(1)$ represents that b has not yet fired the propagation rule from above and $b(0)$ that b has already fired the propagation rule from above. The

resulting program can be proven terminating using the additive extension from above, the transformational approach of [PSDS07a, PSDS07b] or the approach of Chapter 7. However, it cannot be proven terminating using the RC for CHR of Chapter 4: for the first rule a needs to be greater than or equal to $b(1)$ and for the second rule $b(1)$ needs to be strictly greater than a.

5.2.3 Single-headed propagation rules

Reconsider the RC for CHR on simpagation rules. In the condition on first-layer propagation, we wish to verify that a multiset decrease on the simpagation rule cannot be undone by propagation on the added CHR constraints of the rule. Therefore, in the presence of multi-headed propagation rules, when using local conditions, we need to require that any constraint, added by propagation on the added constraints of a simpagation rule, has to be strictly smaller than some constraint in the decreasing ranks set of the strict multiset decrease.

The requirement for a strict decrease can however be relaxed when regarding single-headed propagation rules. For such rules there is no requirement for partner constraints, and thus, if a single-headed propagation rule can be applied on an added CHR constraint of a simpagation rule, it can be applied only once. Consider for example the CHR program

$$a(s(X)), a(X), a(X) \Leftrightarrow a(s(X)). \qquad a(s(X)) \Rightarrow a(X).$$

It is a terminating CHR program, however, it cannot be proven terminating using the RC for CHR. The reason is that for the propagation rule, $a(s(t)) \succ a(t)$ for any term t. Under this assumption, we cannot satisfy the condition on the simpagation rule. That is, for any substitution of a term t for the variable X of the simpagation rule, $[\![a(t)]\!]$ is a decreasing ranks set for the multiset decrease condition of the simpagation rule. Considering first-layer propagation on the added constraint $a(s(t))$ of the simpagation rule, then constraints $a(t)$ are added by the propagation rule.

However, since the propagation rule is single-headed, if a constraint $a(s(t))$ is added by the simpagation rule, the propagation rule can only fire once on the added constraint, resulting in only one addition of a constraint $a(t)$. Considering this, further propagation on $a(t)$ cannot undo the multiset decrease. Unfortunately, the RC for CHR is not sufficiently precise to detect this.

Note that programs like the one above are difficult termination problems. They cannot be handled by a transformation of single-headed propagation rules to simplification rules, combined with either the RC for CHR, the additive extension or multiplicative extension from above, the transformational approach of [PSDS07a, PSDS07b] or the approach of Chapter 7.

Therefore, to handle programs like the one above, we will need to consider the effect of single-headed propagation separately. That is, for single-headed propagation we will need to consider not only the first-layer propagation but also the effect of propagation of further layers. This can be achieved through a refinement of the RC for CHR, however, still requires further investigation.

5.2.4 Refined control

Most operational CHR semantics in practical implementations of CHR are a refinement of the theoretical CHR semantics (see [SVWSDK10]). Since, in general, the reason for a refined semantics is efficiency of execution and not improved control (unless the refinement concerns rule order), it will be so that many CHR programs that terminate for some more refined semantics will also terminate for the theoretical semantics. Therefore, in these cases, a proof of termination can be obtained by application of the RC for CHR of Chapter 4.

In case a refined semantics allows the programmer to consider rule order, the RC for CHR will in many cases be unable to prove termination since programmers tend to use this property. Consider for example the CHR program

$$R_1 \ @ \ a(X, X) \Leftrightarrow true.$$
$$R_2 \ @ \ a(X, Y) \Leftrightarrow a(X, X).$$

The program, executed under the theoretical CHR semantics is not terminating and thus the RC for CHR cannot prove termination. Considering a refinement of the theoretical CHR semantics with rule order, then, since $a/2$ constraints with arguments of the same size are removed before they can be used by the second rule, the program is terminating.

In many cases, if termination depends on rule order, a transformation, based on adding guards to CHR rules, can be applied to obtain the same operational behaviour without rule order. Consider for example the CHR program

$$R_1 \ @ \ a(X, X) \Leftrightarrow true.$$
$$R_2 \ @ \ a(X, Y) \Leftrightarrow X \backslash == Y \mid a(X, X).$$

Executing this program under the theoretical CHR semantics is operational equivalent to executing the program from above, considering rule order. By the guard of the second rule, we prevent that constraints $a(t, t)$ can match the head of the simplification rule. Now, using the RC for CHR, we can prove termination.

A transformational approach on the basis of adding guards is however not always applicable. In particular, this is the case when rule order is used to implement a test for absence of constraints. Such a test has the form

$$R_1 \ @ \ a \setminus apresent(A) \Leftrightarrow A = yes.$$
$$R_2 \ @ \ apresent(A) \Leftrightarrow A = no.$$

By rule order, we are guaranteed that if a constraint a is present, the first rule fires. If not, the second rule fires. Without rule order, in case of absence, we correctly derive absence. However, in case of presence, we can either derive presence or absence. So, if a program's termination depends on such a test, we are not be able to prove termination since a guard cannot replace a test for absence of constraints. One way of handling such problems is to make the number of constraints in the constraint store of some predicate explicit:

$$R_1 \ @ \ a \setminus apresent(A) \Leftrightarrow A = yes.$$
$$R_2 \ @ \ nrofa(0) \setminus apresent(A) \Leftrightarrow A = no.$$

The behaviour of this program under the theoretical CHR semantics is operationally equivalent to the original program with rule order. Note, however, that if the test is part of a larger CHR program, for operational equivalence, we also need to adapt the values of the constraints keeping track of occurrences, whenever related constraints are removed or added. E.g., a rule $(a \Leftrightarrow true.)$ would need to become a rule $(nrofa(s(A)), a \Leftrightarrow nrofa(A).)$.

Finally, a much more difficult refinement to handle are selection rules, e.g., like the selection rule of the refined operational semantics of CHR [DSGH04]. Since in practice, programmers tend to (ab)use selection rules for refined control they often lead to programming errors as —in the presence of multi-headed CHR rules— their effect is often hard to predict. Termination analysis for CHR with selection rules is therefore an interesting, but difficult, problem. Similarly, the use of dynamic rule priorities like in [DK09], where rule order can change during program execution, is an interesting, but difficult, problem.

Part II

Practice

For Manh Thang Nguyen.

Chapter 6

Automating termination analysis for CHR

Similar to LP, in order to prove termination of a CHR program automatically, we require conditions that can be verified in a finite number of steps [DSD94, DSS02]. In general, such conditions are derived from theoretical termination conditions, which typically require an infinite number of verifications. Finally, based on these verifiable conditions, several techniques can be conceived to automate their verification.

For LP, much work has been done for automating termination proofs, e.g., [DDSV99, GC05, SDS03, GTSKF04, SDS04, SDS05b, SDS05a, MB05, BCG+07, GSKT+07, EWZ08, SKGN10, NDSGSK11]. Traditionally, in LP, termination proofs are automated by means of *norms* and *level mappings*, mapping the terms and atoms of the program to the set of natural numbers. Therefore, by the standard ordering on natural numbers, a reduction pair can be induced on the atoms of the program, with which the termination conditions can be verified. Consequently, proving termination is based on the search for a suitable norm and level mapping such that the termination conditions can be satisfied.

Most termination analysis techniques for LP are based on *linear norms* and *level mappings*. These measure the size of each term or atom as a positive linear combination of the sizes of their sub-terms. One example is the *Hasta-La-Vista* system [SDS03]. It infers one specific linear norm and level mapping and in the context of numerical computations, based on a case analysis, it includes a

refinement of this. Other examples are the tool *cTI* [MB05] which uses concrete linear norms and *TermiLog* [LSS97], combining several linear norms together.

As demonstrated in [NDSGSK11], the restriction to linear norms and level mappings limits the power of the analysis considerably. Therefore, in [NDSGSK11], a framework for automated termination analysis with polynomial interpretations has been developed for LP. This work has been inspired on the application of polynomial interpretations in TRS (first introduced by Lankford in [Lan79]), where it is currently one of the best known and most widely used techniques for proving termination. As it turned out, the application of polynomial interpretations to termination analysis of LP resulted in one of the most powerful LP termination analysis techniques yet [NDSGSK11].

Although the approach of [NDSGSK11] is successful on most termination problems of the LP Termination Database [Ter10], it can only handle programs that terminate due to *bounded decrease*. For example, the CHR program $(a(s(X)) \Leftrightarrow a(X).)$ is a program terminating due to bounded decrease. To prove termination of such a program, it is sufficient to verify that the atoms involved in computations can only become "simpler" (i.e., smaller).

For programs terminating due to *bounded increase*, such as a CHR program $(a(X,Y) \Leftrightarrow less(X,Y) \mid a(s(X),Y).)$ (see Example 6.1.1), the atoms involved in computations become more "complex". Hence, [NDSGSK11] and most other approaches —based on norms and level mappings that are some positive combination of the sizes of sub-terms— cannot prove termination. However, in programs like the one above, the "complexity" of these terms or atoms is bounded from above. Therefore, decreases can be verified against these bounds instead (see Example 6.1.1).

To derive these bounds, [GTSKF04] and [SDS03] apply a *domain analysis*. We, in contrast to [GTSKF04] and [SDS03], propose an integrated approach. Our approach extends the approach of [NDSGSK11]. Next to polynomials over the natural numbers, that can only handle bounded decrease, we also allow for polynomials over the integers to handle bounded increase. Furthermore, with our approach, we can also prove termination of programs that only terminate for specific kinds of queries, like for the Merge-sort program of Example 2.2.1 (see Section 2.4.2), as illustrated in Section 6.5.

In this chapter, first, we formulate a verifiable RC for CHR with integer polynomial interpretations. The form of this RC is similar to the termination conditions of [NDSGSK11]. Next, when automating the verification process by a constraint-based approach [DDSV99,NDSGSK11], we only consider linear polynomials. This does not mean that there is any restriction imposed on the use of higher-order polynomials, but we point the reader to [NDSGSK11] for

an in-depth discussion thereof. Finally, note that we consider Prolog as the host language for CHR and that we will prove termination of the Prolog part of a CHR(Prolog) program as well.

6.1 A verifiable RC for CHR

In [PDS09a], we adapted the approach of [NDSGSK11] to CHR. There, it was shown that many of the concepts of [NDSGSK11] are directly applicable to the CHR context. We do not repeat the discussion of [PDS09a] here. Instead, in this chapter, we discuss the approach of [PDS09b], extending the approach of [PDS09a] (and [NDSGSK11]) by considering polynomials over the integers. The following simple example motivates the use of such polynomials.

Example 6.1.1 (motivating example)**.** *Consider the CHR(Prolog) program*

$$R \ @ \ a(X,Y) \Leftrightarrow less(X,Y) \mid a(s(X),Y).$$

$$less(0,s(\_)).$$
$$less(s(X),s(Y)) :- less(X,Y).$$

It is a typical bounded increase problem. The CHR rule R fires on constraints $a(x,y)$ if $P \models less(x,y)$, i.e., x has a strictly smaller term-size than y. Whenever R is applied, it replaces a constraint $a(x,y)$ by a constraint $a(s(x),y)$. The program above is obviously terminating. At some point in the computation, the term-size of the first argument of the $a/2$ constraints will no longer be strictly smaller than the term-size of its second argument.

The approach of [PDS09a], adapted from [NDSGSK11], is not able to handle such problems. This is because [NDSGSK11] and [PDS09a] interpret the sizes of constraints and terms as a positive combination of the sizes of their subterms. Using such interpretations, we can never show a decrease between the constraint used in the head of R and the constraint added in the body of R.

In fact, to verify a decrease, we would need to consider polynomials with integer coefficients: we can verify a decrease between the head and the body of R by subtracting the term-size of the first argument of the $a/2$ constraints from the term-size of their second argument. This difference, as imposed by the guard of R, is always greater than 0 for $a/2$ constraints of the call set. As such, we obtain a strict partial ordering that is well-founded on the call set and for which R demonstrates strict decreases between its head and body. □

Note that in general, to prove termination of LP or CHR programs, the use of polynomials over the integers is not allowed. Using the polynomial ordering,

which we discuss in the next section (see Definition 6.1.1), integer polynomials are not guaranteed to induce a reduction pair on the atoms and terms of the program as required by the RCs of the previous chapters (see Chapters 2, 3 and 4). More specifically, the strict partial ordering induced on the atoms and terms of the program is not guaranteed to be well-founded.

The underlying reason for this is that integer polynomials do not have the property of $\mathbb{N}$-*closedness*. That is, for variable assignments from $\mathbb{N}$, integer polynomials do not necessarily evaluate to numbers in $\mathbb{N}$. Therefore, in the approach of [NDSGSK11], a choice has been made to restrict to polynomials over the natural numbers because such polynomials are $\mathbb{N}$-closed by default. Then, the strict partial ordering induced on the atoms and terms of the program is always well-founded.

To allow for integer polynomials, we will need to impose auxiliary conditions that guarantee $\mathbb{N}$-closedness of integer polynomials. However, we will not require them to be $\mathbb{N}$-closed for all possible assignments from $\mathbb{N}$. Otherwise, it would restrict us to polynomials in $\mathbb{N}$ anyway. Instead, we require $\mathbb{N}$-closedness only for assignments from $\mathbb{N}$ that correspond to constraints of the call set. After all, in the RC for CHR (see Section 4.3.2), the strict partial ordering on the atoms of the program needs only to be well-founded on the call set.

Note that the refinement (of $\mathbb{N}$-closedness) to assignments of the call set, in our approach, is the reason that we, compared to [PDS09a] and [NDSGSK11], obtain a strictly more powerful framework for termination analysis. If in our conditions for $\mathbb{N}$-closedness, we assume the call set to be the set of all atoms of the program, then only polynomials over the natural numbers are allowed. In that case, our approach is equivalent to the approach of [PDS09a]. Otherwise, by a restricted call set, we may be allowed to use integer polynomials as well.

In the remainder of this chapter, we explain our approach by means of the following CHR(Prolog) program. It is an interesting program as with current techniques it can only be proven terminating using the approach of Chapter 4, automated using an approach based on integer polynomial interpretations.

Example 6.1.2. *Consider the CHR(Prolog) program P:*

$$R_1 \ @ \ a(N_1, M), b(N_2, M) \Rightarrow less(N_1, N_2), less(N_2, M) \mid b(s(N_2), M).$$
$$R_2 \ @ \ a(N_1, M), b(N_2, M) \Leftrightarrow less(N_1, M), less(N_2, M) \mid a(N_2, M).$$

$$less(0, s(\_)).$$
$$less(s(X), s(Y)) :- less(X, Y).$$

It is a simplified version of a genetic algorithm. The program is queried with a population of one Alpha, represented as an $a/2$ constraint, and a number

of Betas, represented as $b/2$ constraints. The first argument of the $a/2$ and $b/2$ constraints represent the generation of the Alphas and Betas. The second represents the maximal number of generations that are considered. Note that for reasons of simplicity, we have left out arguments for gene representation and built-ins for crossover of genes and fitness of genes.

The first rule of the program is a reproduction rule: if there is an Alpha and a Beta and the Beta is of a later generation than the Alpha, then a new Beta of a next generation is added once. The second rule represents natural selection: a Beta can become an Alpha. Finally, notice that the reproduction rule and the selection rule can only be applied up till a given number of generations. □

6.1.1 Polynomial interpretations

We say that a variable X occurs in a polynomial p if the polynomial contains a monomial with a coefficient different from 0 and X occurs in this monomial. If $X_1, \ldots, X_n$ are all the variables occurring in a polynomial p, we often denote p as $p(X_1, \ldots, X_n)$. For every polynomial p, there is an associated polynomial function $F_p = \lambda X_1, \ldots, X_n.p(X_1, \ldots, X_n)$. For numbers or polynomials $x_1, \ldots, x_n$, we often write $p(x_1, \ldots, x_n)$ instead of $F_p(x_1, \ldots, x_n)$. Given $p(X_1, \ldots, X_n)$ and $m \geq 1$ we also have an associated polynomial function $F_{p,m} = \lambda X_1, \ldots, X_n, Y_1, \ldots, Y_m.p(X_1, \ldots, X_n)$. For such an associated function on an extended domain, we often write $p(x_1, \ldots, x_n, y_1, \ldots, y_m)$ instead of writing $F_{p,m}(x_1, \ldots, x_n, y_1, \ldots, y_m)$.

By $\Pi_{\mathbb{Z}}$, we represent the set of polynomials over $\mathbb{Z}$, that is polynomials that take coefficients in $\mathbb{Z}$, and by $\Pi_{\mathbb{N}}$ the set of polynomials over $\mathbb{N}$. Obviously, $\Pi_{\mathbb{N}} \subseteq \Pi_{\mathbb{Z}}$. Next, we define an ordering on polynomials of $\Pi_{\mathbb{Z}}$ on the basis of their associated polynomial functions.

Definition 6.1.1 (polynomial ordering on $\Pi_{\mathbb{Z}}$)**.** *Let p and q be two polynomials in $\Pi_{\mathbb{Z}}$. Let $X_1, \ldots, X_n$ be all variables occurring in p and q. Then, $p \succeq_{\mathbb{Z}} q$ iff $\forall x_1, \ldots, x_n \in \mathbb{N} : p(x_1, \ldots, x_n) \geq q(x_1, \ldots, x_n)$.* □

The polynomial ordering $\succeq_{\mathbb{Z}}$ is a partial ordering on $\Pi_{\mathbb{Z}}$, with associated strict partial ordering $\succ_{\mathbb{Z}}$. In contrast to [NDSGSK11], $\succ_{\mathbb{Z}}$ is not defined separately. This is in line with the reduction pairs of the previous chapters.

In the next example, we demonstrate the polynomial ordering.

Example 6.1.3 (polynomial ordering)**.** *Consider the polynomial $p = 4 \cdot X + 3 \cdot Y$ and the polynomial $q = 2 \cdot X$. For their associated polynomial functions holds that $\forall x, y \in \mathbb{N} : 4 \cdot x + 3 \cdot y \geq 2 \cdot x$, thus $p \succeq_{\mathbb{Z}} q$. Furthermore, $q \not\succeq_{\mathbb{Z}} p$, thus,*

$p \succ_{\mathbb{Z}} q$. Finally, $p \succeq_{\mathbb{Z}} 0$ and $q \succeq_{\mathbb{Z}} 0$ since both are in $\Pi_{\mathbb{N}}$. For $p = 4 \cdot X - Y$ and $q = 4 \cdot X$, we have that $q \succ_{\mathbb{Z}} p$. □

Note that in [NDSGSK11], although $p \succeq'_{\mathbb{Z}} q$ is defined as in Definition 6.1.1, that $p \succ'_{\mathbb{Z}} q$ is defined as $\forall x_1, \ldots, x_n \in \mathbb{N} : p(x_1, \ldots, x_n) > q(x_1, \ldots, x_n)$. Therefore, as can be verified, $\succ'_{\mathbb{Z}} \subseteq \succ_{\mathbb{Z}}$. This, however, will not affect the precision of the analysis, as we will discuss in Example 6.1.2.

Finally, note that $\succ_{\mathbb{Z}}$ is not well-founded on $\Pi_{\mathbb{Z}}$, but that it is on $\Pi_{\mathbb{N}}$. To see this, consider the following properties of polynomials in $\Pi_{\mathbb{N}}$.

Proposition 6.1.1. *Let p and q be two polynomials in $\Pi_{\mathbb{N}}$. Furthermore, let $var(p)$ represent the set of variables in p, $sc(p)$ the sum of the coefficients of p and $deg(p)$ the degree of p. If $p \succeq_{\mathbb{Z}} q$, then*

1. *$var(q) \subseteq var(p)$,*
2. *$sc(p) \geq sc(q)$,*
3. *$deg(p) \geq deg(q)$ and*
4. *$\{q \in \Pi_{\mathbb{N}} \mid p \succeq_{\mathbb{Z}} q\}$ is finite.* □

Proof. The proofs of properties 1, 2 and 3 are by contradiction. The proof of property 4 follows from the properties 1, 2 and 3.

1. Otherwise, by assigning 1 to all variables in $var(p)$, p evaluates to $sc(p)$ and q to a polynomial q'. Since q' cannot be a constant polynomial, there must exist an assignment to the variables of q', such that it evaluates to a number strictly greater than $sc(p)$. This contradicts the given.
2. Otherwise, by assigning 1 to all variables in $var(p) \cup var(q)$, p evaluates to $sc(p)$, q to $sc(q)$ and $sc(q) > sc(p)$. This contradicts the given.
3. Otherwise, there is a monomial in q that is of a strictly larger degree than all monomials in p. Consequently, there is an assignment $x_1, \ldots, x_n$ of sufficiently large values such that $q(x_1, \ldots, x_n) > p(x_1, \ldots, x_n)$. This contradicts the given.
4. Property 1, 2 and 3 guarantee that if $p \succeq_{\mathbb{Z}} q$ there can only be a finite number of different polynomials satisfying the condition that $var(q) \subseteq var(p)$, $sc(p) \geq sc(q)$ and $deg(p) \geq deg(q)$ in $\Pi_{\mathbb{N}}$.

□

Considering Property 4 of Proposition 6.1.2, it is obvious that $\succ_{\mathbb{Z}}$ is well-founded on $\Pi_{\mathbb{N}}$.

Recall that it is our objective to define a reduction pair on the call set of a CHR program with which we can satisfy the RC for CHR (see Section 4.3.2). We will do so by a mapping of the constraints of the program to elements of $\Pi_{\mathbb{Z}}$ and compare these constraints on the basis of the polynomial ordering discussed above. To obtain such a mapping, we use a polynomial interpretation in $\Pi_{\mathbb{Z}}$.

Definition 6.1.2 (polynomial interpretation in $\Pi_{\mathbb{Z}}$)**.** *A polynomial interpretation $\mathbb{P}$ for a CHR(Prolog) program P maps each symbol f of arity n in $Const_P \cup Fun_P$ to a polynomial $p_f(X_1, \ldots, X_n) \in \Pi_{\mathbb{N}}$ and maps each symbol c of arity m in $Pred_P$ to a polynomial $p_c(X_1, \ldots, X_m) \in \Pi_{\mathbb{Z}}$.* □

Note that we only map predicate symbols to polynomials in $\Pi_{\mathbb{Z}}$ and functor symbols only to polynomials in $\Pi_{\mathbb{N}}$. This restriction is motivated as follows.

We wish to reuse *polynomial interargument relations* —successfully used in [NDSGSK11] for representing *success set relations*— to express, for CHR, *success* and *call set relations*. These interargument relations can only be used to represent the relations that hold between the arguments of atoms of a predicate. They cannot represent relations that hold between arguments of functors used in atoms of a predicate. Therefore, in order to guarantee well-foundedness of the ordering induced on the call set, the polynomials associated to functor symbols would need to be in $\Pi_{\mathbb{N}}$ anyway (see Section 6.1.2).

We give an example of a polynomial interpretation. First note that we handle the constants of a CHR program by considering them functors of arity 0.

Example 6.1.4 (polynomial interpretation)**.** *Consider the program P of Example 6.1.2, where $Fun_P = \{s/1, 0/0\}$ and $Pred_P = \{a/2, b/2, less/2\}$. Then,*

$$\begin{array}{lll} \mathbb{P}(s/1) = 1 + X_1 & \mathbb{P}(0/0) = 0 & \\ \mathbb{P}(a/2) = X_2 & \mathbb{P}(b/2) = X_2 - X_1 & \mathbb{P}(less/2) = X_1 \end{array}$$

is a possible polynomial interpretation for the symbols of P. □

Every polynomial interpretation induces a norm and a level mapping.

Definition 6.1.3 (polynomial norm and level mapping)**.** *The norm associated with a polynomial interpretation $\mathbb{P}$ is a mapping $\|.\|_{\mathbb{P}} : Term_P \to \Pi_{\mathbb{N}}$, defined:*

- $\|X\|_{\mathbb{P}} = X$ *if X is a variable and*

- $\|f(t_1,\ldots,t_n)\|_{\mathbb{P}} = p_f(\|t_1\|_{\mathbb{P}},\ldots,\|t_n\|_{\mathbb{P}})$, *where* $p_f = \mathbb{P}(f)$.

The level mapping associated with a polynomial interpretation $\mathbb{P}$ *is a mapping* $|.|_{\mathbb{P}} : Atom_P \to \Pi_{\mathbb{Z}}$, *defined in terms of norms:*

- $|c(t_1,\ldots,t_n)|_{\mathbb{P}} = p_c(\|t_1\|_{\mathbb{P}},\ldots,\|t_n\|_{\mathbb{P}})$, *where* $p_c = \mathbb{P}(c)$. □

We illustrate this using the polynomial interpretation of Example 6.1.4.

Example 6.1.5 (polynomial norm and level mapping)**.** *Consider the constraints* $a(s(s(0)), s(s(s(0))))$, $b(s(0), s(s(s(0))))$ *and* $less(s(X), s(s(X)))$ *in* $Atom_P$, *with* P *the CHR(Prolog) program of Example 6.1.2. Furthermore, consider the polynomial interpretation* $\mathbb{P}$ *of Example 6.1.4 for* P. *Then,*

- $|a(s(s(0)), s(s(s(0))))|_{\mathbb{P}} = \|s(s(s(0)))\|_{\mathbb{P}} = 3$,
- $|b(s(0), s(s(s(0))))|_{\mathbb{P}} = (\|s(s(s(0)))\|_{\mathbb{P}}) - (\|s(0)\|_{\mathbb{P}}) = 3 - 1 = 2$ *and*
- $|less(s(X), s(s(X)))|_{\mathbb{P}} = \|s(X)\|_{\mathbb{P}} = 1 + X$. □

By a polynomial norm and level mapping, polynomial interpretations induce corresponding orderings on terms and atoms.

Definition 6.1.4 (ordering on $Term_P \cup Atom_P$ w.r.t. $\mathbb{P}$)**.** *Let* $\mathbb{P}$ *be a polynomial interpretation for a CHR(Prolog) program* P. *We define the binary relation* $\succeq_{\mathbb{P}}$ *on* $Term_P \cup Atom_P$ *as follows:* $s \succeq_{\mathbb{P}} t$ *iff*

- $\|s\|_{\mathbb{P}} \succeq_{\mathbb{Z}} \|t\|_{\mathbb{P}}$ *for* $s, t \in Term_P$,
- $\|s\|_{\mathbb{P}} \succeq_{\mathbb{Z}} |t|_{\mathbb{P}}$ *for* $s \in Term_P$ *and* $t \in Atom_P$,
- $|s|_{\mathbb{P}} \succeq_{\mathbb{Z}} \|t\|_{\mathbb{P}}$ *for* $s \in Atom_P$ *and* $t \in Term_P$, *or*
- $|s|_{\mathbb{P}} \succeq_{\mathbb{Z}} |t|_{\mathbb{P}}$ *for* $s, t \in Atom_P$. □

It can easily be verified that the strict partial ordering $\succ_{\mathbb{P}}$ on $Term_P \cup Atom_P$, associated to $\succeq_{\mathbb{P}}$ as usual, can also be associated to $\succ_{\mathbb{Z}}$ in the same way as $\succeq_{\mathbb{P}}$ is associated to $\succeq_{\mathbb{Z}}$ in Definition 6.1.4. Notice that $\succeq_{\mathbb{P}}$ is not a partial ordering like the polynomial ordering $\succeq_{\mathbb{Z}}$ from which it is induced. This is because syntactically different terms and constraints can be mapped to the same polynomial. Therefore, $\succeq_{\mathbb{P}}$ is a pre-order on $Term_P \cup Atom_P$.

The next example illustrates the ordering on the atoms and terms of the program w.r.t. a polynomial interpretation.

Example 6.1.6 (ordering on $Term_P \cup Atom_P$ w.r.t. $\mathbb{P}$)**.** *Consider the CHR(Prolog) program P of Example 6.1.2 and the polynomial interpretation $\mathbb{P}$ of Example 6.1.4 for P. Then, $|a(s(s(0)), s(s(s(0))))|_{\mathbb{P}} = 3$ and $|b(s(0), s(s(s(0))))|_{\mathbb{P}} = 2$, as in Example 6.1.5. Applying R_1 of P on these constraints results in the addition of $b(s(s(0)), s(s(s(0))))$, where $|b(s(s(0)), s(s(s(0))))|_{\mathbb{P}} = 1$. As can be verified, $a(s(s(0)), s(s(s(0)))) \succ_{\mathbb{P}} b(s(s(0)), s(s(s(0))))$ and $b(s(0), s(s(s(0)))) \succ_{\mathbb{P}} b(s(s(0)), s(s(s(0))))$.*

Now consider the head $less(s(X), s(Y))$ of the Prolog clause of P. Its polynomial level mapping is $1 + X$. Considering the body $less(X, Y)$ of this clause, its polynomial level mapping is X. Obviously, $1 + X \succ_{\mathbb{Z}} X$. Therefore, $less(s(X), s(Y)) \succ_{\mathbb{P}} less(X, Y)$. Furthermore, notice that for every substitution σ of the variables X and Y with terms in $Term_P$ holds that $less(s(X), s(Y))\sigma \succ_{\mathbb{P}} less(X, Y)\sigma$. □

6.1.2 The RC for CHR with polynomial interpretations

We now restate Definition 4.2.3, 4.3.7 and 4.3.8 of the RC for CHR for the special case of polynomial interpretations. That is, we represent the success set and the call set of a CHR program by polynomial interargument relations and formulate rigidity of the call set w.r.t. polynomial interpretations (cfr. [NDSGSK11]). In contrast to [NDSGSK11], we need to verify that the ordering induced on the constraints of the call set by an integer polynomial interpretation yields a reduction pair on the call set and do so by formulating $\mathbb{N}$-closedness of a polynomial interpretation w.r.t. the call set.

Interargument relations for the success set

In order to obtain a termination criterion that is suitable for automation, usually one estimates the effect of intermediate calls by suitable interargument relations. Although this notion can be defined for arbitrary orderings [DSS02], we only introduce *polynomial interargument relations* [NDSGSK11].

Definition 6.1.5 (polynomial interargument relation)**.** *Let $\mathbb{P}$ be a polynomial interpretation for a CHR(Prolog) program P and let c/n be a predicate in P. Then, $\mathfrak{R}_{c/n} = \{c(t_1, \ldots, t_n) \mid \forall i : t_i \in Term_P \wedge p_c^{in}(\|t_1\|_{\mathbb{P}}, \ldots, \|t_n\|_{\mathbb{P}}) \succeq_{\mathbb{Z}} p_c^{out}(\|t_1\|_{\mathbb{P}}, \ldots, \|t_n\|_{\mathbb{P}})$, with $p_c^{in}, p_c^{out} \in \Pi_{\mathbb{N}}\}$ is a* polynomial interargument relation *for c/n.* □

By using interargument relations we can represent the success set of a predicate. Recall that we will only do so for built-in predicates (see Section 2.3.5).

Definition 6.1.6 (success set relation)**.** *Let P be a CHR(Prolog) program, c/n a predicate in P and $\mathbb{P}$ a polynomial interpretation for P. Then, $\mathfrak{R}^{SS}_{c/n}$ is a* success set relation*, if $\mathfrak{R}^{SS}_{c/n}$ is a polynomial interargument relation and $P \models c(t_1,\ldots,t_n) \rightarrow c(t_1,\ldots,t_n) \in \mathfrak{R}^{SS}_{c/n}$.* □

Note that success set relations give a safe over-approximation of the *extended* success set of a predicate (see Definition 2.3.15).

Example 6.1.7 (success set relation)**.** *Consider the CHR(Prolog) program P of Example 6.1.2. Then, for the polynomial interpretation $\mathbb{P}$ of Example 6.1.4, $\mathfrak{R}^{SS}_{less/2} = \{less(t_1,t_2) \mid t_1,t_2 \in Term_P \wedge \|t_2\|_{\mathbb{P}} \succ_{\mathbb{Z}} \|t_1\|_{\mathbb{P}}\}$ is a success set relation for the $less/2$ predicate. E.g., for $less(s(0),s(s(X)))$ in the extended success set of $less/2$: $\|s(0)\|_{\mathbb{P}} = 1$, $\|s(s(X))\|_{\mathbb{P}} = 2 + X$ and $2 + X \succ_{\mathbb{Z}} 1$.* □

Interargument relations for the call set

In contrast to [NDSGSK11], since we allow for polynomials over $\mathbb{Z}$, we need to verify well-foundedness of the ordering induced on the call set. For that purpose, we represent the call set by interargument relations as well.

Definition 6.1.7 (call set relation)**.** *Let P be a CHR(Prolog) program, I a query set for P, c/n a predicate in P and $\mathbb{P}$ a polynomial interpretation for P. Then, $\mathfrak{R}^{CS}_{c/n}$ is a* call set relation*, if $\mathfrak{R}^{CS}_{c/n}$ is a polynomial interargument relation and $C \in Call(P,I) \rightarrow C \in \mathfrak{R}^{CS}_{rel(C)}$.* □

Example 6.1.8 (call set relation)**.** *Consider the CHR(Prolog) program P of Example 6.1.2. Then, for the polynomial interpretation $\mathbb{P}$ of Example 6.1.4, $\mathfrak{R}^{CS}_{a/2} = \{a(t_1,t_2) \mid t_1,t_2 \in Term_P \wedge \|t_2\|_{\mathbb{P}} \succ_{\mathbb{Z}} \|t_1\|_{\mathbb{P}}\}$ is a call set relation for the $a/2$ predicate. E.g., for $a(s(0),s(s(0)))$ in the call set of $a/2$: $\|s(0)\|_{\mathbb{P}} = 1$, $\|s(s(0))\|_{\mathbb{P}} = 2$ and $2 \succ_{\mathbb{Z}} 1$. Furthermore, as can be verified, $\mathfrak{R}^{CS}_{b/2} = \{b(t_1,t_2) \mid t_1,t_2 \in Term_P \wedge \|t_2\|_{\mathbb{P}} \succ_{\mathbb{Z}} \|t_1\|_{\mathbb{P}}\}$ is a call set relation for the $b/2$ predicate.* □

When inferring call set relations, to obtain more precise results, we need to represent the constraints used to query the program. Therefore, we introduce *query set relations*.

Definition 6.1.8 (query set relation)**.** *Let P be a CHR(Prolog) program, I a query set for P, c/n a predicate in P and $\mathbb{P}$ a polynomial interpretation for P. Then, $\mathfrak{R}^{QS}_{c/n}$ is a* query set relation*, if $\mathfrak{R}^{QS}_{c/n}$ is a polynomial interargument relation and $C \in I \rightarrow C \in \mathfrak{R}^{QS}_{rel(C)}$.* □

Furthermore, for inferring call set relations of CHR predicates —representing the constraints in the constraint store *used* by the rules of a CHR program—, we will also need to represent the constraints *added* to the constraint store in computations of the program. Therefore, we introduce *added set relations*.

Definition 6.1.9 (added set relation)**.** *Let P be a CHR(Prolog) program, I a query set for P, c/n a predicate in P, $\mathbb{P}$ a polynomial interpretation of P and $Add(P, I)$ the set of CHR constraints added to the constraint store in computations of P for I. Then, $\mathfrak{R}^{AS}_{c/n}$ is an* added set relation*, if $\mathfrak{R}^{AS}_{c/n}$ is a polynomial interargument relation and $C \in Add(P, I) \rightarrow C \in \mathfrak{R}^{AS}_{rel(C)}$.* □

Added set relations are only useful for CHR predicates. This is because added set relations for Prolog predicates will not result in improved precision. That is, in LP, since there are no explicit guards and instead of matching there is unification, added sets and call sets coincide.

Furthermore, note that CHR constraints in their query set relation are considered to be in their added set relation. That is, constraints used to query the program are added to the constraint store. Thus, $\mathfrak{R}^{QS}_{rel(C)} \subseteq \mathfrak{R}^{AS}_{rel(C)}$ for any CHR constraint C. Hence, by inclusion of the query set into the added set, the call set is the set of constraints added to the constraint store and *used* by the rules of the program. Therefore, CHR constraints in their call set relation need to be in their added set relation as well. Thus, $\mathfrak{R}^{CS}_{rel(C)} \subseteq \mathfrak{R}^{AS}_{rel(C)}$ for any CHR constraint C. For Prolog predicates, since we only consider a call set, $\mathfrak{R}^{QS}_{rel(C)} \subseteq \mathfrak{R}^{CS}_{rel(C)}$ for any built-in constraint C. In Example 6.1.11, we elaborate on this.

Rigidity of the call set

Instead of rigidity w.r.t. general orders as in Definition 2.3.10, we define it w.r.t. polynomial interpretations.

Definition 6.1.10 (rigidity w.r.t. polynomial interpretations)**.** *A term or constraint $C \in Term_P \cup Atom_P$ is called rigid w.r.t. a polynomial interpretation $\mathbb{P}$ iff A is rigid w.r.t. $(\succeq_{\mathbb{P}}, \succ_{\mathbb{P}})$, i.e., iff $A \approx_{\mathbb{P}} A\sigma$ (or equivalently $|A|_{\mathbb{P}} \approx_{\mathbb{Z}} |A\sigma|_{\mathbb{P}}$) holds for any substitution σ. A set of terms or constraints is rigid w.r.t. $\mathbb{P}$ iff each term or constraint in the set is rigid w.r.t. $\mathbb{P}$.* □

For polynomial interpretations, rigidity can also be characterised in an alternative way using relevant variables.

Definition 6.1.11 (relevant variables)**.** *Let $\mathbb{P}$ be a polynomial interpretation for a CHR(Prolog) program P. Let C be a term or constraint in $Term_P \cup$*

$Atom_P$. Then, a variable X in C is called relevant w.r.t. $\mathbb{P}$ if there exists a substitution $\sigma = \{X \setminus t\}$ of a term t for X such that $C \not\approx_{\mathbb{P}} C\sigma$. □

Example 6.1.9 (relevant variables)**.** *Let $\mathbb{P}$ be the polynomial interpretation of Example 6.1.4 for the CHR(Prolog) program P of Example 6.1.2. Consider a constraint $c = less(s(0), s(s(Y)))$. Then, $|c|_{\mathbb{P}} = 1$. Thus, c has no relevant variables w.r.t. $\mathbb{P}$. Now, consider a constraint $c' = less(s(X), s(s(Y)))$. Then, $|c'|_{\mathbb{P}} = 1 + X$. Thus, X is the only relevant variable of c' w.r.t. $\mathbb{P}$.* □

Proposition 6.1.2 (alternative characterisation of rigidity)**.** *Let $\mathbb{P}$ be a polynomial interpretation and C a term or constraint. Then, C is rigid w.r.t. $\mathbb{P}$ iff C has no relevant variables w.r.t. $\mathbb{P}$.* □

Proof. Obvious from Definitions 6.1.10 and 6.1.11. □

Well-foundedness of $\succ_{\mathbb{P}}$ w.r.t. the call set

To verify well-foundedness of $\succ_{\mathbb{P}}$ w.r.t. the call set, we verify $\mathbb{N}$-closedness of the polynomial interpretation $\mathbb{P}$ w.r.t. the call set.

Definition 6.1.12 ($\mathbb{N}$-closedness of $\mathbb{P}$ w.r.t. $Call(P, I)$)**.** *Let $\mathbb{P}$ be a polynomial interpretation for a CHR(Prolog) program P and let I be a query set for P. Then, $\mathbb{P}$ is $\mathbb{N}$-closed w.r.t. $Call(P, I)$ iff for all C in $Call(P, I)$ holds that $|C|_{\mathbb{P}}$ is $\mathbb{N}$-closed, i.e., $|C|_{\mathbb{P}} \succeq_{\mathbb{Z}} 0$.* □

We have the following properties of $\mathbb{N}$-closed polynomials in $\Pi_{\mathbb{Z}}$.

Proposition 6.1.3. *Let p and q be two $\mathbb{N}$-closed polynomials in $\Pi_{\mathbb{Z}}$. Furthermore, let $var(p)$ represent the set of variables in p, $sc(p)$ the sum of the coefficients of p and $deg(p)$ the degree of p. If $p \succeq_{\mathbb{Z}} q$, then*

1. *$var(q) \subseteq var(p)$,*
2. *$sc(p) \geq sc(q) \geq 0$,*
3. *$deg(p) \geq deg(q)$ and*
4. *$\{q \in \Pi_{\mathbb{Z}} \mid p \succeq_{\mathbb{Z}} q \wedge q \succeq_{\mathbb{Z}} 0\}$ is finite.* □

Proof. The proofs of properties 1, 2 and 3 are by contradiction. The proof of property 4 follows from the properties 1, 2 and 3.

1. Otherwise, by assigning 1 to all variables in $var(p)$, p evaluates to $sc(p)$ and q to a polynomial q'. a) If q' is *not* a constant polynomial, there must exist an assignment to the variables of q' such that it evaluates to a number strictly greater than $sc(p)$ or strictly smaller than 0. This contradicts the given. b) If q' is a constant polynomial, this means that for every monomial $q_i \cdot X_a \cdot \cdots \cdot X_b \cdot Y_c \cdot \cdots \cdot Y_d$ in q containing newly introduced variables —where we use X to denote variables from $var(p)$ and Y variables from $var(q) \setminus var(p)$—, there must exist a second monomial $-q_i \cdot X'_a \cdot \cdots \cdot X'_b \cdot Y_c \cdot \cdots \cdot Y_d$ in q such that $X_a \cdot \cdots \cdot X_b$ is different from $X'_a \cdot \cdots \cdot X'_b$. Otherwise the monomials would have cancelled out. Considering that we can write these pairs of monomials as $q_i \cdot Y_c \cdot \cdots \cdot Y_d \cdot (X_a \cdot \cdots \cdot X_b - X'_a \cdot \cdots \cdot X'_b)$, it is not hard to see that there must exist an assignment to the variables in $var(p)$ —which is different from 1— such that p evaluates to a constant c_p and q to a non-constant polynomial q''. Hence, there must exist an assignment to the variables of q'' such that it evaluates to a number strictly greater than c_p or strictly smaller than 0. This contradicts the given.

2. Otherwise, by assigning 1 to all variables in $var(p) \cup var(q)$, p evaluates to $sc(p)$, q to $sc(q)$ and $sc(q)$ is either strictly greater than $sc(p)$ or strictly smaller than 0. This contradicts the given.

3. Otherwise, there is a monomial in q that is of a strictly larger degree than all monomials in p. Consequently, there is an assignment $x_1, \ldots, x_n$ of sufficiently large values such that $q(x_1, \ldots, x_n) > p(x_1, \ldots, x_n)$ or $0 > q(x_1, \ldots, x_n)$. This contradicts the given.

4. Property 1, 2 and 3 guarantee that if $p \succeq_{\mathbb{Z}} q$ there can only be a finite number of different polynomials satisfying the condition that $var(q) \subseteq var(p)$, $sc(p) \geq sc(q)$ and $deg(p) \geq deg(q)$ in $\Pi_{\mathbb{N}}$.

□

Thus, if a polynomial interpretation is $\mathbb{N}$-closed for the call set, it yields a reduction pair on the call set.

Proposition 6.1.4. *Let P be a CHR(Prolog) program and I a query set for P. If a polynomial interpretation $\mathbb{P}$ for P is $\mathbb{N}$-closed w.r.t. $Call(P, I)$, then $(\succeq_{\mathbb{P}}, \succ_{\mathbb{P}})$ is a reduction pair on $Call(P, I)$.* □

Proof. First, consider that $\succeq_{\mathbb{P}}$ is a pre-order relation on $Atom_P$ and thus also on $Call(P, I) \subseteq Atom_P$. Secondly, by contradiction, assume presence of an infinite strict decreasing chain $C_0 \succ_{\mathbb{P}} C_1 \succ_{\mathbb{P}} \cdots \succ_{\mathbb{P}} C_i \succ_{\mathbb{P}} \ldots$ of constraints C_i

in $Call(P, I)$. Then, corresponding to this sequence, there is an infinite strict decreasing chain of polynomials $|C_0|_{\mathbb{P}} \succ_{\mathbb{Z}} |C_1|_{\mathbb{P}} \succ_{\mathbb{Z}} \cdots \succ_{\mathbb{Z}} |C_i|_{\mathbb{P}} \succ_{\mathbb{Z}} \ldots$ in $\Pi_{\mathbb{Z}}$, where $\forall i : |C_i|_{\mathbb{P}} \succeq_{\mathbb{Z}} 0$ (see Definition 6.1.12). Since all these polynomials must be different, this contradicts Property 4 in Proposition 6.1.3 that states that there can only be a finite number of different polynomials that are smaller than $|C_0|_{\mathbb{P}}$ and greater than 0. □

Note that when considering all atoms of the program, our approach reduces to the approach of [NDSGSK11]. If we use call set relations in a pre-condition, e.g., $(\forall C \in Atom_P : C \in \mathfrak{R}^{CS}_{rel(C)} \rightarrow |C|_{\mathbb{P}} \succeq_{\mathbb{Z}} 0)$, such that we restrict to instantiations corresponding to atoms of the call set, we can be allowed to use integer polynomials as well. We provide an example.

Example 6.1.10 ($\mathbb{N}$-closedness w.r.t. $Call(P, I)$)**.** *Consider the CHR(Prolog) program P and its intended use I as given in Example 6.1.2. Furthermore, consider the polynomial interpretation $\mathbb{P}$ for P as given in Example 6.1.4. Finally, consider the call set relation $\mathfrak{R}^{CS}_{a/2}$ for the $a/2$ predicate as given in Example 6.1.8. Then, $\forall a(t_1, t_2) \in Atom_P : a(t_1, t_2) \in \mathfrak{R}^{CS}_{a/2} \rightarrow |a(t_1, t_2)|_{\mathbb{P}} \succeq_{\mathbb{Z}} 0$. Reworking this condition, we obtain: $\forall t_1, t_2 \in Term_P : \|t_2\|_{\mathbb{P}} \succ_{\mathbb{Z}} \|t_1\|_{\mathbb{P}} \rightarrow \|t_2\|_{\mathbb{P}} \succeq_{\mathbb{Z}} 0$; which is obviously true. Note that since $\mathbb{P}(a/2) \in \Pi_{\mathbb{N}}$, there is no need to verify $\mathbb{N}$-closedness as such polynomials are $\mathbb{N}$-closed by default.*

Now consider the call set relation $\mathfrak{R}^{CS}_{b/2}$ for the $b/2$ predicate as given in Example 6.1.8. Then, $\forall b(t_1, t_2) \in Atom_P : b(t_1, t_2) \in \mathfrak{R}^{CS}_{b/2} \rightarrow |b(t_1, t_2)|_{\mathbb{P}} \succeq_{\mathbb{Z}} 0$. Reworking this condition, we obtain: $\forall t_1, t_2 \in Term_P : \|t_2\|_{\mathbb{P}} \succ_{\mathbb{Z}} \|t_1\|_{\mathbb{P}} \rightarrow \|t_2\|_{\mathbb{P}} - \|t_1\|_{\mathbb{P}} \succeq_{\mathbb{Z}} 0$; which is easily verified to be true.

Since the interpretation of the other symbols of the program are in $\Pi_{\mathbb{N}}$, we can conclude that $\mathbb{P}$ is $\mathbb{N}$-closed w.r.t. $Call(P, I)$ and therefore that $(\succeq_{\mathbb{P}}, \succ_{\mathbb{P}})$ is a reduction pair on $Call(P, I)$. □

The RC for CHR with polynomial interpretations

In the RC for CHR with polynomial interpretations, compared to the RC for CHR (see Chapter 4), we additionally require in the decrease condition on simpagation rules that the strict multiset decrease has a singleton decreasing ranks sets (see Definitions 4.3.6 and 4.3.7). Furthermore, in the decrease conditions for propagation and simpagation rules, we consider all the body constraints and not only those which under substitution belong to the call set. This is not a problem since $\succeq_{\mathbb{P}}$ is an ordering on $Term_P \cup Atom_P$ and not $Call(P, I)$ alone. In practice, these choices do not seem to affect precision much (see Chapter 8), but greatly reduce the complexity of the analysis (see

Section 6.2.5). Note however that, in general, the *answer constraints* of the program are ignored (see Section 8.1).

Although in this chapter we do not assume termination of the Prolog part of the program, we will still restrict attention to CHR programs where built-in constraints can only result in variable bindings. Therefore, we will leave the built-in constraints of the body of CHR rules unconsidered in the decrease conditions of the RC for CHR with polynomial interpretations.

Definition 6.1.13 (RC for CHR with polynomial interpretations)**.** *Let P be a CHR(Prolog) program and I a query set. Let $\mathbb{P}$ be a polynomial interpretation for P and $\succeq_{\mathbb{P}}$ the ordering induced on $Term_P \cup Atom_P$ w.r.t. $\mathbb{P}$. Then, a CHR program P for I satisfies the* RC for CHR with polynomial interpretations *iff:*

- *For each built-in and CHR predicate c/n there is a call set relation $\mathfrak{R}^{CS}_{c/n}$ in P w.r.t. $\mathbb{P}$ and for each built-in predicate c/n there is a success set relation $\mathfrak{R}^{SS}_{c/n}$ in P w.r.t. $\mathbb{P}$.*
- *$(\succeq_{\mathbb{P}}, \succ_{\mathbb{P}})$ is a reduction pair on $Call(P, I)$, with $\succ_{\mathbb{P}}$ the strict partial order associated to $\succeq_{\mathbb{P}}$, such that $Call(P, I)$ is rigid w.r.t. $(\succeq_{\mathbb{P}}, \succ_{\mathbb{P}})$.*
- *For any propagation rule*

$$R_p \ @ \ H_1, \ldots, H_n \Rightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

  in P and for any substitution χ such that

$$\forall H \in [\![H_1, \ldots, H_n]\!]\chi : H \in \mathfrak{R}^{CS}_{rel(H)},$$
$$\forall G \in [\![G_1, \ldots, G_k]\!]\chi : G \in \mathfrak{R}^{SS}_{rel(G)} \text{ and}$$
$$\forall B \in [\![B_1, \ldots, B_l]\!]\chi : B \in \mathfrak{R}^{SS}_{rel(B)},$$

  holds that $\forall H \in [\![H_1, \ldots, H_n]\!]\chi$ and $\forall C \in [\![C_1, \ldots, C_m]\!]\chi$: $H \succ_{\mathbb{P}} C$.
- *Let $(\succeq_{\mu_{\mathbb{P}}}, \succ_{\mu_{\mathbb{P}}})$ be the multiset extension of $(\succeq_{\mathbb{P}}, \succ_{\mathbb{P}})$.*
  For any simpagation rule

$$R_s \ @ \ H_1, \ldots, H_j \setminus H_{j+1}, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m.$$

  in P and for any substitution χ such that

$$\forall H \in [\![H_1, \ldots, H_n]\!]\chi : H \in \mathfrak{R}^{CS}_{rel(H)},$$
$$\forall G \in [\![G_1, \ldots, G_k]\!]\chi : G \in \mathfrak{R}^{SS}_{rel(G)} \text{ and}$$
$$\forall B \in [\![B_1, \ldots, B_l]\!]\chi : B \in \mathfrak{R}^{SS}_{rel(B)},$$

holds that $[\![H_{j+1}, \ldots, H_n]\!]\chi \succ_{\mu_{\mathbb{P}}} [\![C_1, \ldots, C_m]\!]\chi$, *with a singleton decreasing ranks set* $[\![\rho]\!]$.

Furthermore, for any propagation rule

$$R_p \ @ \ H'_1, \ldots, H'_{n'} \Rightarrow G'_1, \ldots, G'_{k'} \mid B'_1, \ldots, B'_{l'}, C'_1, \ldots, C'_{m'}.$$

in P *and for any substitution* χ', *such that*

$$\forall H' \in [\![H'_1, \ldots, H'_{n'}]\!]\chi' : H' \in \mathfrak{R}^{CS}_{rel(H)},$$
$$\forall G' \in [\![G'_1, \ldots, G'_{k'}]\!]\chi' : G' \in \mathfrak{R}^{SS}_{rel(G')} \text{ and}$$
$$\forall B' \in [\![B'_1, \ldots, B'_{l'}]\!]\chi' : B' \in \mathfrak{R}^{SS}_{rel(B')},$$

and such that there exists a constraint D:

$$D \in [\![C_1, \ldots, C_m]\!]\chi \text{ and } D \in [\![H'_1, \ldots, H'_{n'}]\!]\chi';$$

holds that $\forall C' \in [\![C'_1, \ldots, C'_{m'}]\!]\chi' \colon \rho \succ C'$. □

Thus, if a polynomial interpretation can be found such that the RC for CHR with polynomial interpretations is satisfied, we prove termination, provided that the host language is terminating as well.

Note that since the host language cannot introduce CHR constraints, we can prove termination of the Prolog part of the program separately. Therefore, in the RC for CHR with polynomial interpretations, we did not include conditions for termination of the Prolog part of the program. In fact, to prove termination of the Prolog part of the program, we can reuse the conditions of [NDSGSK11], considering that an extension to integer polynomial interpretations of [NDSGSK11] requires a verification of $\mathbb{N}$-closedness of the Prolog predicates of the program. In Section 6.2, we make these conditions for Prolog explicit.

Corollary 6.1.1. *If the RC for CHR with polynomial interpretations is satisfied for a CHR(Prolog) program* P *with query set* I, *then* P *is terminating for* I. □

Proof. The corollary immediately follows from Definitions 4.2.3, 4.3.7 and 4.3.8, and Theorem 4.3.1. □

A procedure for verifying termination of CHR(Prolog) programs

Corollary 6.1.1, a specialisation of Theorem 4.3.1, can be applied to verify termination of a CHR(Prolog) program P with a query set I. In practice, we

will also verify termination of the Prolog part of a CHR(Prolog) program, using the approach of [NDSGSK11] extended to integer polynomial interpretations. Thus, to prove termination of a CHR(Prolog) program, we have to check that all conditions in the following termination proof procedure are satisfied by some polynomial interpretation $\mathbb{P}$.

Procedure 6.1.1. *Let P be a CHR(Prolog) program, I a query set and $\mathbb{P}$ a polynomial interpretation of the symbols of P. Then, the termination proof procedure derived from Corollary 6.1.1 contains the following 5 steps:*

- **Step 1***: For all built-in predicates of the program we must infer success set relations.*
- **Step 2***: For all built-in and CHR predicates of the program we must infer call set relations.*
- **Step 3***: $Call(P, I)$ must be rigid w.r.t. $\mathbb{P}$. Thus, no atom C in $Call(P, I)$ may have relevant variables w.r.t. $\mathbb{P}$.*
- **Step 4***: $(\succeq_{\mathbb{P}}, \succ_{\mathbb{P}})$ must be a reduction pair on $Call(P, I)$. Thus, given the call set relations inferred in Step 2, $\mathbb{P}$ must be $\mathbb{N}$-closed w.r.t. $Call(P, I)$.*
- **Step 5***: Given the success and call set relations inferred in Step 1 and 2, for every Prolog clause, the decrease conditions of [NDSGSK11] must be satisfiable, and for every CHR rule, the decrease conditions of the RC for CHR with polynomial interpretations (from Definition 6.1.13) must be satisfiable.* □

For **Step 1**, like in [NDSGSK11], we follow the standard approach in LP to verify that a relation holds for all elements of the Herbrand model (see [Llo87]). That is, first we verify that a success set relation holds for the facts of the CHR(Prolog) program. Then, we verify for all possible instantiations of a clause such that all body atoms are in their success set relations, whether the head atom is also in its success set relation.

For **Step 2**, by a top-down analysis instead, call set relations can be inferred similar to success set relations in **Step 1**. That is, for the Prolog part of the program, we verify for all possible instantiations of a clause whether a body atom is in its call set relation, given that the head atom is in its call set relation and that the intermediate body atoms —considering a left-to-right selection rule for Prolog— are in their success set relations.

For the CHR part of the CHR(Prolog) program, *first*, we verify for all possible instantiations of a CHR rule whether a guard is in its call set relation, given that the head constraints are in their call set relations and that the intermediate

guards —considering a left-to-right selection rule for Prolog— are in their success set relations. *Secondly*, we verify for all possible instantiations of a CHR rule whether a body built-in constraint is in its call set relation, given that the head constraints are in their call set relations and that all guards are in their success set relations. *Thirdly*, we verify for all possible instantiations of a CHR rule whether a body CHR constraint is in its *added* set relation, given that the head constraints are in their call set relations and that the guards are in their success set relations. *Finally*, we relate added set relations to call set relations. That is, we verify for all possible instantiations of a CHR rule whether a head constraint is in its call set relation, given that the head constraint is in its added set relation and that all guards are in their success set relations. In **Step 2** of the example below (see Example 6.1.11), we explain this in more detail. There, we also discuss the use of query set relations.

For **Step 3**, in general, we derive modes or call types to obtain information on the relevant variables of atoms in the call set. An extensive discussion on the use of modes and call types for rigidity can be found in [Ngu09]. To verify rigidity, the approach of [NDSGSK11] uses call type information [JB92] to obtain conditions on the coefficients of a polynomial interpretation, derived on the basis of *critical paths* in *rigid call graphs* (see [NDSGSK11] and [JB92]). In [PDS09a], this approach is successfully adapted to the CHR context.

In this text, for reasons of simplicity, we avoid discussion of techniques for verifying rigidity by restricting to ground CHR(Prolog), i.e., CHR(Prolog) programs in which every call to the program is ground. For such programs there cannot be relevant variables. Hence, we have rigidity by default. Moreover, as explained in [PDS09a] and [NDSGSK11], the extension to non-ground programs is straightforward anyway.

Example 6.1.11 (applying Corollary 6.1.1 to Example 6.1.2)**.** *Consider the CHR(Prolog) program P of Example 6.1.2:*

$$R_1 \ @ \ a(N_1, M), b(N_2, M) \Rightarrow less(N_1, N_2), less(N_2, M) \mid b(s(N_2), M).$$
$$R_2 \ @ \ a(N_1, M), b(N_2, M) \Leftrightarrow less(N_1, M), less(N_2, M) \mid a(N_2, M).$$

$$less(0, s(\_)).$$
$$less(s(X), s(Y)) :- less(X, Y).$$

Furthermore, consider the polynomial interpretation $\mathbb{P}$ for P of Example 6.1.4:

$$\mathbb{P}(s/1) = 1 + X_1 \quad \mathbb{P}(0/0) = 0$$
$$\mathbb{P}(a/2) = X_2 \quad \mathbb{P}(b/2) = X_2 - X_1 \quad \mathbb{P}(less/2) = X_1$$

To prove termination of P for I, we apply Procedure 6.1.1:

Step 1: *We may infer the following success set relation for* $less/2$*:*

$\mathfrak{R}^{SS}_{less/2} = \{less(t_1, t_2) \mid t_1, t_2 \in Term_P \wedge \|t_2\|_{\mathbb{P}} \succ_{\mathbb{Z}} \|t_1\|_{\mathbb{P}}\}$.

Checking that $\mathfrak{R}^{SS}_{less/2}$ *is indeed a success set relation for* $less/2$ *is equivalent to verifying the correctness of the conditions:*

$\forall X \in Term_P : less(0, s(X)) \in \mathfrak{R}^{SS}_{less/2}$ *and*
$\forall X, Y \in Term_P : less(X, Y) \in \mathfrak{R}^{SS}_{less/2} \rightarrow less(s(X), s(Y)) \in \mathfrak{R}^{SS}_{less/2}$.

Or equivalently (see Definition 6.1.1),

$\forall X \in \mathbb{N} : 1 + X > 0$ *and*
$\forall X, Y \in \mathbb{N} : Y > X \rightarrow 1 + Y > 1 + X$.

These conditions are obviously true.

Step 2: *If the intended use of* P *is given by the query set relations:*

$\mathfrak{R}^{QS}_{a/2} = \{a(t_1, t_2) \mid t_1, t_2 \in Term_P \wedge \|t_2\|_{\mathbb{P}} \succ_{\mathbb{Z}} \|t_1\|_{\mathbb{P}}\}$,
$\mathfrak{R}^{QS}_{b/2} = \{b(t_1, t_2) \mid t_1, t_2 \in Term_P \wedge \|t_2\|_{\mathbb{P}} \succ_{\mathbb{Z}} \|t_1\|_{\mathbb{P}}\}$ *and*
$\mathfrak{R}^{QS}_{less/2} = \{less(t_1, t_2) \mid t_1, t_2 \in Term_P \wedge false\} = \emptyset$;

then, we may infer the call set relations:

$\mathfrak{R}^{CS}_{a/2} = \{a(t_1, t_2) \mid t_1, t_2 \in Term_P \wedge \|t_2\|_{\mathbb{P}} \succ_{\mathbb{Z}} \|t_1\|_{\mathbb{P}}\}$,
$\mathfrak{R}^{CS}_{b/2} = \{b(t_1, t_2) \mid t_1, t_2 \in Term_P \wedge \|t_2\|_{\mathbb{P}} \succ_{\mathbb{Z}} \|t_1\|_{\mathbb{P}}\}$ *and*
$\mathfrak{R}^{CS}_{less/2} = \{less(t_1, t_2) \mid t_1, t_2 \in Term_P \wedge true\}$;

and the added set relations:

$\mathfrak{R}^{AS}_{a/2} = \{a(t_1, t_2) \mid t_1, t_2 \in Term_P \wedge \|t_2\|_{\mathbb{P}} \succ_{\mathbb{Z}} \|t_1\|_{\mathbb{P}}\}$ *and*
$\mathfrak{R}^{AS}_{b/2} = \{b(t_1, t_2) \mid t_1, t_2 \in Term_P \wedge \|t_2\|_{\mathbb{P}} \succeq_{\mathbb{Z}} \|t_1\|_{\mathbb{P}}\}$.

Recall that there is no use in differentiating between call set relations and added set relations of Prolog predicates. Therefore, for Prolog predicates, we only consider query and call set relations. Furthermore, note that the relations $false$ *and* $true$ *can be represented by the interargument relations* $0 \succ_{\mathbb{Z}} 1$ *and* $1 \succ_{\mathbb{Z}} 0$*, respectively.*

Checking that $\mathfrak{R}^{CS}_{a/2}$, $\mathfrak{R}^{CS}_{b/2}$ *and* $\mathfrak{R}^{CS}_{less/2}$ *are indeed call set relations for the predicates* $a/2$, $b/2$ *and* $less/2$*, respectively, and that* $\mathfrak{R}^{AS}_{a/2}$ *and* $\mathfrak{R}^{AS}_{b/2}$ *are indeed added set relations for the predicates* $a/2$ *and* $b/2$*, respectively, is equivalent to verifying the correctness of the following conditions.*

For the first CHR rule, we need to verify that

$\forall N_1, N_2, M \in Term_P : a(N_1, M) \in \mathfrak{R}^{CS}_{a/2} \wedge b(N_2, M) \in \mathfrak{R}^{CS}_{b/2} \wedge$
$less(N_1, N_2) \in \mathfrak{R}^{SS}_{less/2} \wedge less(N_2, M) \in \mathfrak{R}^{SS}_{less/2} \rightarrow$
$b(s(N_2), M) \in \mathfrak{R}^{AS}_{b/2}$.

That is, if the constraints used in the head of the first rule are part of their call set relations and if the guards are in their success set relations,

then the body CHR constraints must be in their added set relations. Equivalently, we need to verify that

$$\forall N_1, N_2, M \in \mathbb{N}: \\ M > N_1 \wedge M > N_2 \wedge N_2 > N_1 \wedge M > N_2 \rightarrow M \geq 1 + N_2.$$

Similarly, for the second CHR rule, we need to verify that

$$\forall N_1, N_2, M \in Term_P : a(N_1, M) \in \mathfrak{R}^{CS}_{a/2} \wedge b(N_2, M) \in \mathfrak{R}^{CS}_{b/2} \wedge \\ less(N_1, M) \in \mathfrak{R}^{SS}_{less/2} \wedge less(N_2, M) \in \mathfrak{R}^{SS}_{less/2} \rightarrow a(N_2, M) \in \mathfrak{R}^{AS}_{a/2}$$

or equivalently that

$$\forall N_1, N_2, M \in \mathbb{N}: \\ M > N_1 \wedge M > N_2 \wedge M > N_1 \wedge M > N_2 \rightarrow M > N_2.$$

Furthermore, for the first CHR rule, we have to verify that

$$\forall N_1, N_2, M \in Term_P : a(N_1, M) \in \mathfrak{R}^{AS}_{a/2} \wedge \\ less(N_1, N_2) \in \mathfrak{R}^{SS}_{less/2} \wedge less(N_2, M) \in \mathfrak{R}^{SS}_{less/2} \rightarrow a(N_1, M) \in \mathfrak{R}^{CS}_{a/2}$$

and that

$$\forall N_1, N_2, M \in Term_P : b(N_2, M) \in \mathfrak{R}^{AS}_{b/2} \wedge \\ less(N_1, N_2) \in \mathfrak{R}^{SS}_{less/2} \wedge less(N_2, M) \in \mathfrak{R}^{SS}_{less/2} \rightarrow b(N_2, M) \in \mathfrak{R}^{CS}_{b/2}.$$

That is, if a constraint in its added set relation can be used to fire the first rule, then it must be in its call set relation. Equivalently, we need to verify that

$$\forall N_1, N_2, M \in \mathbb{N} : M > N_1 \wedge N_2 > N_1 \wedge M > N_2 \rightarrow M > N_1$$

and that

$$\forall N_1, N_2, M \in \mathbb{N} : M \geq N_2 \wedge N_2 > N_1 \wedge M > N_2 \rightarrow M > N_2.$$

For the second CHR rule, we have to verify that

$$\forall N_1, N_2, M \in Term_P : a(N_1, M) \in \mathfrak{R}^{AS}_{a/2} \wedge \\ less(N_1, M) \in \mathfrak{R}^{SS}_{less/2} \wedge less(N_2, M) \in \mathfrak{R}^{SS}_{less/2} \rightarrow a(N_1, M) \in \mathfrak{R}^{CS}_{a/2}$$

and that

$$\forall N_1, N_2, M \in Term_P : b(N_2, M) \in \mathfrak{R}^{AS}_{b/2} \wedge \\ less(N_1, M) \in \mathfrak{R}^{SS}_{less/2} \wedge less(N_2, M) \in \mathfrak{R}^{SS}_{less/2} \rightarrow b(N_2, M) \in \mathfrak{R}^{CS}_{b/2}.$$

Equivalently, we need to verify that

$$\forall N_1, N_2, M \in \mathbb{N} : M > N_1 \wedge M > N_1 \wedge M > N_2 \rightarrow M > N_1$$

and that

$$\forall N_1, N_2, M \in \mathbb{N} : M \geq N_2 \wedge M > N_1 \wedge M > N_2 \rightarrow M > N_2.$$

Since we will also prove termination of the Prolog part of a CHR(Prolog) program, we consider call set relations for built-in predicates. Therefore, for the Prolog clause of the program, we have to verify that

$$\forall X, Y \in Term_P : less(s(X), s(Y)) \in \mathfrak{R}^{CS}_{less/2} \rightarrow less(X, Y) \in \mathfrak{R}^{CS}_{less/2}.$$

That is, if a built-in constraint that can be used to fire a Prolog clause is in its call set relation and all of the intermediate calls are in their success set relations, then the added body built-in constraints must be in their call set relations. Equivalently, we need to verify that

$$\forall X, Y \in \mathbb{N} : true \rightarrow true.$$

Furthermore, we will also need to verify for the first CHR rule that

$$\forall N_1, N_2, M \in Term_P : a(N_1, M) \in \mathfrak{R}^{CS}_{a/2} \wedge b(N_2, M) \in \mathfrak{R}^{CS}_{b/2} \rightarrow less(N_1, N_2) \in \mathfrak{R}^{CS}_{less/2}.$$
$$\forall N_1, N_2, M \in Term_P : a(N_1, M) \in \mathfrak{R}^{CS}_{a/2} \wedge b(N_2, M) \in \mathfrak{R}^{CS}_{b/2} \wedge less(N_1, N_2) \in \mathfrak{R}^{SS}_{less/2} \rightarrow less(N_2, M) \in \mathfrak{R}^{CS}_{less/2}.$$

That is, for every guard of the first CHR rule, if the constraints used to fire the first CHR rule are part of their call set relations and if the intermediate guards are part of their success set relations, then the considered guard needs to be part of its call set relation. Equivalently, we need to verify that

$$\forall N_1, N_2, M \in \mathbb{N} : M > N_1 \wedge M > N_2 \rightarrow true$$
$$\forall N_1, N_2, M \in \mathbb{N} : M > N_1 \wedge M > N_2 \wedge N_2 > N_1 \rightarrow true.$$

For the second CHR rule, we need to verify that

$$\forall N_1, N_2, M \in Term_P : a(N_1, M) \in \mathfrak{R}^{CS}_{a/2} \wedge b(N_2, M) \in \mathfrak{R}^{CS}_{b/2} \rightarrow less(N_1, M) \in \mathfrak{R}^{CS}_{less/2}.$$
$$\forall N_1, N_2, M \in Term_P : a(N_1, M) \in \mathfrak{R}^{CS}_{a/2} \wedge b(N_2, M) \in \mathfrak{R}^{CS}_{b/2} \wedge less(N_1, M) \in \mathfrak{R}^{SS}_{less/2} \rightarrow less(N_2, M) \in \mathfrak{R}^{CS}_{less/2}.$$

Equivalently, we need to verify that

$$\forall N_1, N_2, M \in \mathbb{N} : M > N_1 \wedge M > N_2 \rightarrow true$$
$$\forall N_1, N_2, M \in \mathbb{N} : M > N_1 \wedge M > N_2 \wedge M > N_1 \rightarrow true.$$

Finally, for the CHR predicates, we relate query set relations to added set relations. We have

$$\forall N, M \in Term_P : a(N, M) \in \mathfrak{R}^{QS}_{a/2} \rightarrow a(N, M) \in \mathfrak{R}^{AS}_{a/2} \text{ and}$$
$$\forall N, M \in Term_P : b(N, M) \in \mathfrak{R}^{QS}_{b/2} \rightarrow b(N, M) \in \mathfrak{R}^{AS}_{b/2},$$

or equivalently,

$$\forall N, M \in \mathbb{N} : M > N \rightarrow M > N \text{ and}$$
$$\forall N, M \in \mathbb{N} : M > N \rightarrow M \geq N.$$

For Prolog predicates, we relate query set relations directly to call set relations. We have

$$\forall X, Y \in Term_P : less(X, Y) \in \mathfrak{R}^{QS}_{less/2} \rightarrow less(X, Y) \in \mathfrak{R}^{CS}_{less/2},$$

or equivalently,

$\forall X, Y \in \mathbb{N} : false \rightarrow true.$

Correctness of the above conditions is easily verified. Note that in Section 6.2.2, we discuss symbolic conditions for deriving call set relations for a CHR(Prolog) program. In that context, we will express these symbolic conditions directly in terms of inequalities over $\mathbb{N}$.

Step 3*: Since we only consider ground CHR(Prolog) there is no need to verify whether there are* no *relevant variables w.r.t.* $\mathbb{P}$. *This is so by default. In general, however, as in [PDS09a] and [NDSGSK11], we would rely on a call type analysis, such as the one of [JB92] and made available to CHR by the approach of [PSB10]. Then, we would use the approach of [NDSGSK11] based on* critical paths. *Such an approach yields constraints, directly expressed on the coefficients of the polynomial interpretation.*

Note that the information represented by call set relations is complementary to the information represented by modes or call types. That is, modes and call types express information on the parts of constraints that will be instantiated in calls to the program, while call set relations express information on the relations that hold across these instantiated parts.

Step 4*: To verify that* $(\succeq_{\mathbb{P}}, \succ_{\mathbb{P}})$ *is a reduction pair on* $Call(P, I)$, *we verify that* $\mathbb{P}$ *is* $\mathbb{N}$*-closed w.r.t.* $Call(P, I)$. *That is,*

$$\forall N, M \in Term_P : a(N, M) \in \mathfrak{R}^{CS}_{a/2} \rightarrow |a(N, M)|_{\mathbb{P}} \succeq_{\mathbb{Z}} 0,$$
$$\forall N, M \in Term_P : b(N, M) \in \mathfrak{R}^{CS}_{b/2} \rightarrow |b(N, M)|_{\mathbb{P}} \succeq_{\mathbb{Z}} 0 \text{ and}$$
$$\forall X, Y \in Term_P : less(X, Y) \in \mathfrak{R}^{CS}_{less/2} \rightarrow |less(X, Y)|_{\mathbb{P}} \succeq_{\mathbb{Z}} 0.$$

The first and third condition are trivially satisfied since $\mathbb{P}(a/2) \in \Pi_{\mathbb{N}}$ *and* $\mathbb{P}(less/2) \in \Pi_{\mathbb{N}}$. *Verifying correctness of the second condition, is equivalent to verifying that* $\forall N, M \in \mathbb{N} : M > N \rightarrow M - N \geq 0$. *Since this is obviously the case,* $(\succeq_{\mathbb{P}}, \succ_{\mathbb{P}})$ *is a reduction pair on* $Call(P, I)$.

Step 5*: Finally, we need to verify satisfaction of the decrease conditions of Corollary 6.1.1. First, we verify satisfaction of the decrease conditions resulting from the propagation rule of* P. *That is, we need to verify that*

$$\forall N_1, N_2, M \in Term_P : a(N_1, M) \in \mathfrak{R}^{CS}_{a/2} \wedge b(N_2, M) \in \mathfrak{R}^{CS}_{b/2} \wedge$$
$$less(N_1, N_2) \in \mathfrak{R}^{SS}_{less/2} \wedge less(N_2, M) \in \mathfrak{R}^{SS}_{less/2} \rightarrow$$
$$a(N_1, M) \succ_{\mathbb{P}} b(s(N_2), M)$$

and that

$$\forall N_1, N_2, M \in Term_P : a(N_1, M) \in \mathfrak{R}^{CS}_{a/2} \wedge b(N_2, M) \in \mathfrak{R}^{CS}_{b/2} \wedge$$
$$less(N_1, N_2) \in \mathfrak{R}^{SS}_{less/2} \wedge less(N_2, M) \in \mathfrak{R}^{SS}_{less/2} \rightarrow$$
$$b(N_2, M) \succ_{\mathbb{P}} b(s(N_2), M).$$

Or equivalently, verifying correctness of

$$\forall N_1, N_2, M \in \mathbb{N} : M > N_1 \wedge M > N_2 \wedge N_2 > N_1 \wedge M > N_2 \rightarrow M > M - (1 + N_2)$$

and

$$\forall N_1, N_2, M \in \mathbb{N} : M > N_1 \wedge M > N_2 \wedge N_2 > N_1 \wedge M > N_2 \rightarrow M - N_2 > M - (1 + N_2).$$

These conditions can easily be verified to be true.

For the simpagation rule of P, we need to verify that

$$\forall N_1, N_2, M \in Term_P : a(N_1, M) \in \mathfrak{R}^{CS}_{a/2} \wedge b(N_2, M) \in \mathfrak{R}^{CS}_{b/2} \wedge less(N_1, M) \in \mathfrak{R}^{SS}_{less/2} \wedge less(N_2, M) \in \mathfrak{R}^{SS}_{less/2} \rightarrow [\![a(N_1, M), b(N_2, M)]\!] \succ_{\mu_{\mathbb{P}}} [\![a(N_2, M)]\!]$$

with decreasing ranks set $[\![\rho]\!]$. Furthermore, we have to verify that

$$\forall N_1, N_2, M, N_1', N_2', M' \in Term_P : a(N_1, M) \in \mathfrak{R}^{CS}_{a/2} \wedge b(N_2, M) \in \mathfrak{R}^{CS}_{b/2} \wedge less(N_1, M) \in \mathfrak{R}^{SS}_{less/2} \wedge less(N_2, M) \in \mathfrak{R}^{SS}_{less/2} \wedge N_2 = N_1' \wedge M = M' \wedge a(N_1', M') \in \mathfrak{R}^{CS}_{a/2} \wedge b(N_2', M') \in \mathfrak{R}^{CS}_{b/2} \wedge less(N_1', N_2') \in \mathfrak{R}^{SS}_{less/2} \wedge less(N_2', M') \in \mathfrak{R}^{SS}_{less/2} \rightarrow \rho \succ_{\mathbb{P}} b(s(N_2'), M').$$

Equivalently, we first need to verify that

$$\forall N_1, N_2, M \in \mathbb{N} : M > N_1 \wedge M > N_2 \wedge M > N_1 \wedge M > N_2 \rightarrow [\![M, M - N_2]\!] >_{\mu} [\![M]\!].$$

This is indeed the case since M appears on both sides of the multiset decrease. Furthermore, it can then be verified that the decreasing ranks set, $[\![M - N_2]\!]$, is singleton. Then, given $M - N_2$, for the decrease condition on first-layer propagation we have to verify correctness of

$$\forall N_1, N_2, M, N_1', N_2', M' \in \mathbb{N} : M > N_1 \wedge M > N_2 \wedge M > N_1 \wedge M > N_2 \wedge N_2 = N_1' \wedge M = M' \wedge M' > N_1' \wedge M' > N_2' \wedge N_2' > N_1' \wedge M' > N_2' \rightarrow M - N_2 > M' - (1 + N_2').$$

Considering that $N_2 = N_1' \wedge M = M' \wedge N_2' > N_1'$, this condition is true.

We also verify conditions for termination of the Prolog part of the program. Based on the approach of [NDSGSK11], we need to verify that $\forall X, Y \in Term_P : less(s(X), s(Y)) \in \mathfrak{R}^{CS}_{less/2} \rightarrow less(s(X), s(Y)) \succ_{\mathbb{P}} less(X, Y)$, or equivalently, that $\forall X, Y \in \mathbb{N} : true \rightarrow 1 + X > X$, which can easily be verified to be true. Note that in Section 6.2, we make the conditions of [NDSGSK11] explicit.

Conclusion: *We may conclude that the CHR(Prolog) program P for its intended use I is terminating.* □

In the next section, we automate Procedure 6.1.1 for verifying termination of a CHR(Prolog) program. That is, we will automate the search for a polynomial interpretation with which all the conditions for termination, resulting from Procedure 6.1.1, can be satisfied.

6.2 Automating the termination proof

To automate the search for a polynomial interpretation that satisfies all of the criteria of Corollary 6.1.1, stated in Procedure 6.1.1, we continue in light of the constraint-based approach of [DDSV99], further improved in [NDSGSK11]. That is, instead of selecting a particular interpretation and interargument relations, we introduce a generic symbolic form for the polynomials that can be used in a polynomial interpretation and interargument relations. Then, we reformulate the conditions for termination using the symbolic polynomials and, finally, reformulate these conditions as constraints on the symbolic coefficients of the symbolic polynomials. If a solution exists to the constraint problem on symbolic coefficients, then this solution yields a concrete polynomial interpretation and concrete polynomial interargument relations with which we prove termination.

So our approach for termination analysis of CHR(Prolog) works as follows:

1. Introduce symbolic versions of the polynomials associated with constant, functor and predicate symbols.
2. Introduce symbolic versions of the polynomial inequalities associated with query, call, added and success set relations.
3. Express all conditions of Procedure 6.1.1, resulting from Corollary 6.1.1, and transform them to constraints on symbolic coefficients.
4. Solve the resulting constraints to obtain values for the coefficients.

Note that there may be multiple solutions for the constraints on symbolic coefficients and that each solution gives rise to a concrete polynomial interpretation and concrete polynomial interargument relations with which we prove termination.

In order to assign symbolic polynomials to the functor and predicate symbols, we restrict to classes of polynomials. Here, we only discuss the *linear class*.

Definition 6.2.1 (linear polynomial class)**.** *Each monomial of a polynomial of the* linear polynomial class *contains at most one variable of at most degree 1:* $p(X_1, \ldots, X_n) = \Sigma_{k=1}^{n} p_k \cdot X_k$. □

In [NDSGSK11] also the *mixed form*, where the monomials of a polynomial may be of a degree 2, is used. This resulted in a considerable increase in expressive power. For reasons of simplicity, we do not discuss symbolic forms for higher-order polynomials although they can be used in our approach as well. For a discussion on their use and applicability, we refer to [NDSGSK11] instead.

Definition 6.2.2 (linear symbolic polynomial interpretation)**.** *A* linear symbolic polynomial interpretation $\mathbb{P}^s$ *for a CHR(Prolog) program P maps each symbol f of arity n in $Const_P \cup Fun_P$ to a linear symbolic polynomial $f_0 + f_1 \cdot X_1 + \cdots + f_n \cdot X_n$ and maps each symbol c of arity m in $Pred_P$ to the difference of two linear symbolic polynomials $c_0^{pos} + c_1^{pos} \cdot X_1 + \cdots + c_n^{pos} \cdot X_m - (c_0^{neg} + c_1^{neg} \cdot X_1 + \cdots + c_n^{neg} \cdot X_m)$.* □

Note that there is a good reason for mapping predicate symbols to the difference of two linear symbolic polynomials. That is, since predicate symbols may be mapped to integer polynomials and since integer polynomials can be represented as the difference of two positive integer polynomials, we only need a constraint solver in $\mathbb{N}$. Otherwise, only for finding values for the symbolic coefficients related to predicate symbols, we would need a solver in $\mathbb{Z}$.

Example 6.2.1 (linear symbolic polynomial interpretation)**.** *Reconsider the CHR(Prolog) program P of Example 6.1.2. Then, we have the following linear symbolic polynomial interpretation $\mathbb{P}^s$ of the symbols in $Fun_P = \{s/1, 0/0\}$ and $Pred_P = \{a/2, b/2, less/2\}$ of P:*

$$\mathbb{P}^s(0/0) = 0_0.$$

$$\mathbb{P}^s(s/1) = s_0 + s_1 \cdot X_1$$

$$\mathbb{P}^s(a/2) = a_0^{pos} + a_1^{pos} \cdot X_1 + a_2^{pos} \cdot X_2 - (a_0^{neg} + a_1^{neg} \cdot X_1 + a_2^{neg} \cdot X_2)$$

$$\mathbb{P}^s(b/2) = b_0^{pos} + b_1^{pos} \cdot X_1 + b_2^{pos} \cdot X_2 - (b_0^{neg} + b_1^{neg} \cdot X_1 + b_2^{neg} \cdot X_2)$$

$$\mathbb{P}^s(less/2) = less_0^{pos} + less_1^{pos} \cdot X_1 + less_2^{pos} \cdot X_2 - (less_0^{neg} + less_1^{neg} \cdot X_1 + less_2^{neg} \cdot X_2)$$ □

Next, we define *symbolic norms* and *level mappings* in terms of a linear symbolic polynomial interpretation.

Definition 6.2.3 (symbolic polynomial norm and level mapping)**.** *The* symbolic norm*, associated with a linear symbolic polynomial interpretation $\mathbb{P}^s$, is defined:*

- $\|X\|_{\mathbb{P}}^s = X$ *if X is a variable and*

- $\|f(t_1,\ldots,t_n)\|^s_{\mathbb{P}} = f_0 + f_1 \cdot \|t_1\|^s_{\mathbb{P}} + \cdots + f_n \cdot \|t_n\|^s_{\mathbb{P}}$.

The symbolic level mapping*, associated with a linear symbolic polynomial interpretation $\mathbb{P}^s$, is defined in terms of norms:*

- $|c(t_1,\ldots,t_n)|^s_{\mathbb{P}} = c_0^{pos} + c_1^{pos} \cdot \|t_1\|^s_{\mathbb{P}} + \cdots + c_n^{pos} \cdot \|t_n\|^s_{\mathbb{P}} - (c_0^{neg} + c_1^{neg} \cdot \|t_1\|^s_{\mathbb{P}} + \cdots + c_n^{neg} \cdot \|t_n\|^s_{\mathbb{P}})$. □

We provide an example.

Example 6.2.2 (symbolic polynomial norm and level mapping)**.** *Consider the linear symbolic polynomial interpretation $\mathbb{P}^s$ of Example 6.2.1. Then, we have for $less(s(X), s(Y))$ that*

$$|less(s(X), s(Y))|^s_{\mathbb{P}} = less_0^{pos} + less_1^{pos} \cdot (s_0 + s_1 \cdot X_1) + less_2^{pos} \cdot (s_0 + s_1 \cdot X_2) - (less_0^{neg} + less_1^{neg} \cdot (s_0 + s_1 \cdot X_1) + less_2^{neg} \cdot (s_0 + s_1 \cdot X_2)).$$ □

We also define *linear symbolic polynomial interargument relations.*

Definition 6.2.4 (linear symbolic polynomial interargument relation)**.** *Let $\mathbb{P}^s$ be a linear symbolic polynomial interpretation for a CHR(Prolog) program P and let c/n be a predicate in P. Then, $\mathfrak{R}^s_{c/n} = \{c(t_1,\ldots,t_n) \mid \forall i : t_i \in Term_P \wedge c_0^{in} + c_1^{in} \cdot \|t_1\|^s_{\mathbb{P}} + \cdots + c_n^{in} \cdot \|t_n\|^s_{\mathbb{P}} \succeq_{\mathbb{Z}} c_0^{out} + c_1^{out} \cdot \|t_1\|^s_{\mathbb{P}} + \cdots + c_n^{out} \cdot \|t_n\|^s_{\mathbb{P}}\}$ is a* linear symbolic polynomial interargument relation *for c/n.* □

Note that we will additionally add superscripts to the symbolic coefficients of linear symbolic interargument relations to make a distinction between success set (SS), call set (CS), added set (AS) and query set relations (QS).

Example 6.2.3 (linear symbolic polynomial interargument relation)**.** *Consider the linear symbolic polynomial interpretation $\mathbb{P}^s$ of Example 6.2.1. Then, we have the following linear symbolic polynomial interargument relation to represent the success set relation of the $less/2$ predicate:*

$$\mathfrak{R}^{SSs}_{less/2} = \{less(t_1,t_2) \mid t_1,t_2 \in Term_P \wedge less_0^{SSin} + less_1^{SSin} \cdot \|t_1\|^s_{\mathbb{P}} + less_2^{SSin} \cdot \|t_2\|^s_{\mathbb{P}} \succeq_{\mathbb{Z}} less_0^{SSout} + less_1^{SSout} \cdot \|t_1\|^s_{\mathbb{P}} + less_2^{SSout} \cdot \|t_2\|^s_{\mathbb{P}}\}.$$ □

In the next subsections, we reformulate the termination conditions of Corollary 6.1.1 (see Procedure 6.1.1) as conditions on symbolic coefficients. This includes inferring success and call set relations (Step 1 and 2), the rigidity property (Step 3), the well-foundedness property (Step 4) and the decrease conditions of the RC for CHR with polynomial interpretations and of [NDSGSK11] (Step 5).

Note that the conditions of the next subsections are directly expressed in terms of inequalities over $\mathbb{N}$ (see Example 6.1.11). Furthermore, note that for any sequence of terms, $t_1, \ldots, t_n$, by $R^s_{c/n}(t_1, \ldots, t_n)$, we abbreviate for a predicate c/n, the inequality $c_0^{in} + c_1^{in} \cdot \|t_1\|^s_{\mathbb{P}} + \cdots + c_n^{in} \cdot \|t_n\|^s_{\mathbb{P}} \geq c_0^{out} + c_1^{out} \cdot \|t_1\|^s_{\mathbb{P}} + \cdots + c_n^{out} \cdot \|t_n\|^s_{\mathbb{P}}$, using superscripts SS, CS, AS and QS when it concerns success, call, added and query set relations, respectively. E.g., $R^{SSs}_{c/n}(t_1, \ldots, t_n)$ abbreviates $c_0^{SSin} + c_1^{SSin} \cdot \|t_1\|^s_{\mathbb{P}} + \cdots + c_n^{SSin} \cdot \|t_n\|^s_{\mathbb{P}} \geq c_0^{SSout} + c_1^{SSout} \cdot \|t_1\|^s_{\mathbb{P}} + \cdots + c_n^{SSout} \cdot \|t_n\|^s_{\mathbb{P}}$. Finally, note that we will also abbreviate sequences of terms, $t_1, \ldots, t_n$, by $\overline{t}$ and sequences of variables, $X_1, \ldots, X_n$, by $\overline{X}$.

6.2.1 Conditions for inferring the success set relations

Since we are only inferring success set relations for Prolog predicates, our approach to infer success set relations is identical to the approach of [NDSGSK11]. For Prolog built-ins, we provide the success set relations as a linear polynomial interargument relation. Thus, the values for the symbolic coefficients of the symbolic success set relations of the Prolog built-in predicates are fixed. For Prolog predicates of the program, we infer success set relations.

To infer success set relations, we formulate conditions on symbolic polynomial interargument relations. That is, for each each Prolog fact $(p(\overline{t}).)$ of a CHR(Prolog) program, there is a condition

$$\forall \overline{X} \in \mathbb{N} : true \rightarrow R_p^{SSs}(\overline{t})$$

and for each Prolog clause $(p(\overline{t}) :- p_1(\overline{t_1}), \ldots, p_m(\overline{t_m}).)$ of a CHR(Prolog) program, there is a condition

$$\forall \overline{X} \in \mathbb{N} : R_{p_1}^{SSs}(\overline{t_1}) \wedge \cdots \wedge R_{p_m}^{SSs}(\overline{t_m}) \rightarrow R_p^{SSs}(\overline{t}).$$

Note that $\overline{X}$ represents the sequence of all variables occurring in the condition. We provide an example.

Example 6.2.4 (symbolic success set conditions)**.** *Recall the Prolog part of the CHR(Prolog) program P of Example 6.1.2:*

$$less(0, s(Y)). \qquad less(s(X), s(Y)) :- less(X, Y).$$

Then, for the success set relation of less/2, we have the symbolic conditions:

$$\forall \overline{X} \in \mathbb{N} : true \rightarrow less_0^{SSin} + less_1^{SSin} \cdot 0_0 + less_2^{SSin} \cdot (s_0 + s_1 \cdot Y) \geq less_0^{SSout} + less_1^{SSout} \cdot 0_0 + less_2^{SSout} \cdot (s_0 + s_1 \cdot Y)$$

$\forall \overline{X} \in \mathbb{N} : less_0^{SSin} + less_1^{SSin} \cdot X + less_2^{SSin} \cdot Y \geq less_0^{SSout} + less_1^{SSout} \cdot X + less_2^{SSout} \cdot Y \rightarrow less_0^{SSin} + less_1^{SSin} \cdot (s_0 + s_1 \cdot X) + less_2^{SSin} \cdot (s_0 + s_1 \cdot Y) \geq less_0^{SSout} + less_1^{SSout} \cdot (s_0 + s_1 \cdot X) + less_2^{SSout} \cdot (s_0 + s_1 \cdot Y)$. □

6.2.2 Conditions for inferring the call set relations

To prove termination of both the Prolog and CHR part of a CHR(Prolog) program, we infer call set relations for both Prolog and CHR predicates. For Prolog predicates, we consider for each Prolog clause $(p(\overline{t}) :- p_1(\overline{t_1}), \ldots, p_m(\overline{t_m}).)$ of a CHR(Prolog) program m conditions:

$$\forall \overline{X} \in \mathbb{N} : R_p^{CSs}(\overline{t}) \rightarrow R_{p_1}^{CSs}(\overline{t_1})$$

$$\forall \overline{X} \in \mathbb{N} : R_p^{CSs}(\overline{t}) \wedge R_{p_1}^{SSs}(\overline{t_1}) \rightarrow R_{p_2}^{CSs}(\overline{t_2})$$

$$\ldots$$

$$\forall \overline{X} \in \mathbb{N} : R_p^{CSs}(\overline{t}) \wedge R_{p_1}^{SSs}(\overline{t_1}) \wedge \cdots \wedge R_{p_{m-1}}^{SSs}(\overline{t_{m-1}}) \rightarrow R_{p_m}^{CSs}(\overline{t_m}).$$

That is, for every body atom of a Prolog clause, if the head of the Prolog clause belongs to its call set relation and, considering a left-to-right selection rule, if all intermediate calls belong to their success set relations, then the body atom must belong to its call set relation.

For inferring call set relations for Prolog and CHR predicates, we consider for each propagation rule

$$h_1(\overline{t_1^h}), \ldots, h_n(\overline{t_n^h}) \Rightarrow g_1(\overline{t_1^g}), \ldots, g_k(\overline{t_k^g}) \mid b_1(\overline{t_1^b}), \ldots, b_l(\overline{t_l^b}), c_1(\overline{t_1^c}), \ldots, c_m(\overline{t_m^c}).$$

and for each simpagation rule

$$h_1(\overline{t_1^h}), \ldots, h_n(\overline{t_n^h}) \Leftrightarrow g_1(\overline{t_1^g}), \ldots, g_k(\overline{t_k^g}) \mid b_1(\overline{t_1^b}), \ldots, b_l(\overline{t_l^b}), c_1(\overline{t_1^c}), \ldots, c_m(\overline{t_m^c}).$$

of a CHR(Prolog) program k conditions:

$$\forall \overline{X} \in \mathbb{N} : R_{h_1}^{CSs}(\overline{t_1^h}) \wedge \cdots \wedge R_{h_n}^{CSs}(\overline{t_n^h}) \rightarrow R_{g_1}^{CSs}(\overline{t_1^g})$$

$$\forall \overline{X} \in \mathbb{N} : R_{h_1}^{CSs}(\overline{t_1^h}) \wedge \cdots \wedge R_{h_n}^{CSs}(\overline{t_n^h}) \wedge R_{g_1}^{SSs}(\overline{t_1^g}) \rightarrow R_{g_2}^{CSs}(\overline{t_2^g})$$

$$\ldots$$

$$\forall \overline{X} \in \mathbb{N} : R_{h_1}^{CSs}(\overline{t_1^h}) \wedge \cdots \wedge R_{h_n}^{CSs}(\overline{t_n^h}) \wedge R_{g_1}^{SSs}(\overline{t_1^g}) \wedge \cdots \wedge R_{g_{k-1}}^{SSs}(\overline{t_{k-1}^g}) \rightarrow R_{g_k}^{CSs}(\overline{t_k^g});$$

l conditions:

$$\forall \overline{X} \in \mathbb{N} : R_{h_1}^{CSs}(\overline{t_1^h}) \wedge \cdots \wedge R_{h_n}^{CSs}(\overline{t_n^h}) \wedge R_{g_1}^{SSs}(\overline{t_1^g}) \wedge \cdots \wedge R_{g_k}^{SSs}(\overline{t_k^g}) \rightarrow R_{b_1}^{CSs}(\overline{t_1^b})$$

$$\forall \overline{X} \in \mathbb{N} : R_{h_1}^{CSs}(\overline{t_1^h}) \wedge \cdots \wedge R_{h_n}^{CSs}(\overline{t_n^h}) \wedge R_{g_1}^{SSs}(\overline{t_1^g}) \wedge \cdots \wedge R_{g_k}^{SSs}(\overline{t_k^g}) \rightarrow R_{b_2}^{CSs}(\overline{t_2^b})$$

$$\ldots$$

$$\forall \overline{X} \in \mathbb{N} : R_{h_1}^{CSs}(\overline{t_1^h}) \wedge \cdots \wedge R_{h_n}^{CSs}(\overline{t_n^h}) \wedge R_{g_1}^{SSs}(\overline{t_1^g}) \wedge \cdots \wedge R_{g_k}^{SSs}(\overline{t_k^g}) \rightarrow R_{b_l}^{CSs}(\overline{t_l^b});$$

and m conditions:

$$\forall \overline{X} \in \mathbb{N} : R_{h_1}^{CSs}(\overline{t_1^h}) \wedge \cdots \wedge R_{h_n}^{CSs}(\overline{t_n^h}) \wedge R_{g_1}^{SSs}(\overline{t_1^g}) \wedge \cdots \wedge R_{g_k}^{SSs}(\overline{t_k^g}) \rightarrow R_{c_1}^{ASs}(\overline{t_1^c})$$

$$\forall \overline{X} \in \mathbb{N} : R_{h_1}^{CSs}(\overline{t_1^h}) \wedge \cdots \wedge R_{h_n}^{CSs}(\overline{t_n^h}) \wedge R_{g_1}^{SSs}(\overline{t_1^g}) \wedge \cdots \wedge R_{g_k}^{SSs}(\overline{t_k^g}) \rightarrow R_{c_2}^{ASs}(\overline{t_2^c})$$

$$\ldots$$

$$\forall \overline{X} \in \mathbb{N} : R_{h_1}^{CSs}(\overline{t_1^h}) \wedge \cdots \wedge R_{h_n}^{CSs}(\overline{t_n^h}) \wedge R_{g_1}^{SSs}(\overline{t_1^g}) \wedge \cdots \wedge R_{g_k}^{SSs}(\overline{t_k^g}) \rightarrow R_{c_m}^{ASs}(\overline{t_m^c}).$$

That is, considering a left-to-right selection rule for Prolog, for each guard in the rule, if all head constraints belong to their call set relations and all intermediate guards belong to their success set relations, then the guard belongs to its call set relation. Furthermore, if all head constraints belong to their call set relations and if all guards belong to their success set relations, then each built-in constraint must belong to its call set relation. Finally, if all head constraints belong to their call set relations and all guards belong to their success set relations, then each body CHR constraint belongs to its added set relation.

Since CHR constraints added in the body of a CHR rule belong to their added set relations (see Example 6.1.11), in CHR, we must relate added sets to call sets as well. Thus, for each head $h(\bar{t})$ of a CHR rule with guards $g_1(\overline{t_1^g}), \ldots, g_k(\overline{t_k^g})$, there is also a condition

$$\forall \overline{X} \in \mathbb{N} : R_h^{ASs}(\bar{t}) \wedge R_{g_1}^{SSs}(\overline{t_1^g}) \wedge \cdots \wedge R_{g_k}^{SSs}(\overline{t_k^g}) \rightarrow R_h^{CSs}(\bar{t}).$$

That is, if a CHR constraint in the added set matches with the head of a CHR rule, such that the guards are all in their success set relations, then that constraint is in its call set relation.

Finally, to handle a description of the intended use by means of query set relations, for Prolog predicates, we need to relate the constraints of the query

set to constraints of the call set. Thus, for each Prolog predicate c/n, there is a condition

$$\forall \overline{X} \in \mathbb{N} : R_c^{QSs}(X_1, \ldots, X_n) \rightarrow R_c^{CSs}(X_1, \ldots, X_n).$$

For CHR predicates, query sets are related to added sets (see Example 6.1.11). Thus, for each CHR predicate c/n, there is a condition

$$\forall \overline{X} \in \mathbb{N} : R_c^{QSs}(X_1, \ldots, X_n) \rightarrow R_c^{ASs}(X_1, \ldots, X_n).$$

Notice that for termination analysis, values for the symbolic coefficients of the query set relations $R_c^{QSs}(X_1, \ldots, X_n)$ need to be fixed. These values are either defined by given query set relations or, in case query set relations are not given, by some relation that will always succeed (i.e., to consider all constraints in $Atom_P$ of a predicate c/n). However, for termination inference (see e.g., [MB05]), the coefficients of symbolic query set relations can be left open.

Example 6.2.5 (symbolic call set conditions)**.** *Consider the CHR(Prolog) program P of Example 6.1.2:*

R_1 @ $a(N_1, M), b(N_2, M) \Rightarrow less(N_1, N_2), less(N_2, M) \mid b(s(N_2), M).$
R_2 @ $a(N_1, M), b(N_2, M) \Leftrightarrow less(N_1, M), less(N_2, M) \mid a(N_2, M).$

$less(0, s(\_)).$
$less(s(X), s(Y)) :- less(X, Y).$

Then, for the call set relations of $a/2$, $b/2$ and $less/2$, we have the following symbolic conditions.

For the guards of the first CHR rule, we have the conditions:

$$\forall \overline{X} \in \mathbb{N} : a_0^{CSin} + a_1^{CSin} \cdot N_1 + a_2^{CSin} \cdot M \geq a_0^{CSout} + a_1^{CSout} \cdot N_1 + a_2^{CSout} \cdot M \wedge b_0^{CSin} + b_1^{CSin} \cdot N_2 + b_2^{CSin} \cdot M \geq b_0^{CSout} + b_1^{CSout} \cdot N_2 + b_2^{CSout} \cdot M \rightarrow less_0^{CSin} + less_1^{CSin} \cdot N_1 + less_2^{CSin} \cdot N_2 \geq less_0^{CSout} + less_1^{CSout} \cdot N_1 + less_2^{CSout} \cdot N_2$$

$$\forall \overline{X} \in \mathbb{N} : a_0^{CSin} + a_1^{CSin} \cdot N_1 + a_2^{CSin} \cdot M \geq a_0^{CSout} + a_1^{CSout} \cdot N_1 + a_2^{CSout} \cdot M \wedge b_0^{CSin} + b_1^{CSin} \cdot N_2 + b_2^{CSin} \cdot M \geq b_0^{CSout} + b_1^{CSout} \cdot N_2 + b_2^{CSout} \cdot M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_1 + less_2^{SSin} \cdot N_2 \geq less_0^{SSout} + less_1^{SSout} \cdot N_1 + less_2^{SSout} \cdot N_2 \rightarrow less_0^{CSin} + less_1^{CSin} \cdot N_2 + less_2^{CSin} \cdot M \geq less_0^{CSout} + less_1^{CSout} \cdot N_2 + less_2^{CSout} \cdot M.$$

For the added CHR constraint of the first CHR rule, we have the condition:

$$\forall \overline{X} \in \mathbb{N} : a_0^{CSin} + a_1^{CSin} \cdot N_1 + a_2^{CSin} \cdot M \geq a_0^{CSout} + a_1^{CSout} \cdot N_1 + a_2^{CSout} \cdot M \wedge b_0^{CSin} + b_1^{CSin} \cdot N_2 + b_2^{CSin} \cdot M \geq b_0^{CSout} + b_1^{CSout} \cdot N_2 + b_2^{CSout} \cdot M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_1 + less_2^{SSin} \cdot N_2 \geq less_0^{SSout} + less_1^{SSout} \cdot N_1 + less_2^{SSout} \cdot N_2 \wedge less_0^{SSin} + less_1^{SSin} \cdot N_2 + less_2^{SSin} \cdot M \geq less_0^{SSout} + less_1^{SSout} \cdot N_2 + less_2^{SSout} \cdot M \rightarrow b_0^{ASin} + b_1^{ASin} \cdot (s_0 + s_1 \cdot N_2) + b_2^{ASin} \cdot M \geq b_0^{ASout} + b_1^{ASout} \cdot (s_0 + s_1 \cdot N_2) + b_2^{ASout} \cdot M.$$

For the guards of the second CHR rule, we have the conditions:

$$\forall \overline{X} \in \mathbb{N} : a_0^{CSin} + a_1^{CSin} \cdot N_1 + a_2^{CSin} \cdot M \geq a_0^{CSout} + a_1^{CSout} \cdot N_1 + a_2^{CSout} \cdot M \wedge b_0^{CSin} + b_1^{CSin} \cdot N_2 + b_2^{CSin} \cdot M \geq b_0^{CSout} + b_1^{CSout} \cdot N_2 + b_2^{CSout} \cdot M \rightarrow less_0^{CSin} + less_1^{CSin} \cdot N_1 + less_2^{CSin} \cdot M \geq less_0^{CSout} + less_1^{CSout} \cdot N_1 + less_2^{CSout} \cdot M$$

$$\forall \overline{X} \in \mathbb{N} : a_0^{CSin} + a_1^{CSin} \cdot N_1 + a_2^{CSin} \cdot M \geq a_0^{CSout} + a_1^{CSout} \cdot N_1 + a_2^{CSout} \cdot M \wedge b_0^{CSin} + b_1^{CSin} \cdot N_2 + b_2^{CSin} \cdot M \geq b_0^{CSout} + b_1^{CSout} \cdot N_2 + b_2^{CSout} \cdot M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_1 + less_2^{SSin} \cdot M \geq less_0^{SSout} + less_1^{SSout} \cdot N_1 + less_2^{SSout} \cdot M \rightarrow less_0^{CSin} + less_1^{CSin} \cdot N_2 + less_2^{CSin} \cdot M \geq less_0^{CSout} + less_1^{CSout} \cdot N_2 + less_2^{CSout} \cdot M.$$

For the added CHR constraint of the second CHR rule, we have the condition:

$$\forall \overline{X} \in \mathbb{N} : a_0^{CSin} + a_1^{CSin} \cdot N_1 + a_2^{CSin} \cdot M \geq a_0^{CSout} + a_1^{CSout} \cdot N_1 + a_2^{CSout} \cdot M \wedge b_0^{CSin} + b_1^{CSin} \cdot N_2 + b_2^{CSin} \cdot M \geq b_0^{CSout} + b_1^{CSout} \cdot N_2 + b_2^{CSout} \cdot M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_1 + less_2^{SSin} \cdot M \geq less_0^{SSout} + less_1^{SSout} \cdot N_1 + less_2^{SSout} \cdot M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_2 + less_2^{SSin} \cdot M \geq less_0^{SSout} + less_1^{SSout} \cdot N_2 + less_2^{SSout} \cdot M \rightarrow a_0^{ASin} + a_1^{ASin} \cdot N_2 + a_2^{ASin} \cdot M \geq a_0^{ASout} + a_1^{ASout} \cdot N_2 + a_2^{ASout} \cdot M.$$

For the Prolog clause, we have the condition:

$$\forall \overline{X} \in \mathbb{N} : less_0^{CSin} + less_1^{CSin} \cdot (s_0 + s_1 \cdot X) + less_2^{CSin} \cdot (s_0 + s_1 \cdot Y) \geq less_0^{CSout} + less_1^{CSout} \cdot (s_0 + s_1 \cdot X) + less_2^{CSout} \cdot (s_0 + s_1 \cdot Y) \rightarrow less_0^{CSin} + less_1^{CSin} \cdot X + less_2^{CSin} \cdot Y \geq less_0^{CSout} + less_1^{CSout} \cdot X + less_2^{CSout} \cdot Y.$$

For the first CHR rule, to relate added sets to call sets, we have the conditions:

$$\forall \overline{X} \in \mathbb{N} : a_0^{ASin} + a_1^{ASin} \cdot N_1 + a_2^{ASin} \cdot M \geq a_0^{ASout} + a_1^{ASout} \cdot N_1 + a_2^{ASout} \cdot M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_1 + less_2^{SSin} \cdot N_2 \geq less_0^{SSout} + less_1^{SSout} \cdot N_1 + less_2^{SSout} \cdot N_2 \wedge less_0^{SSin} + less_1^{SSin} \cdot N_2 + less_2^{SSin} \cdot M \geq less_0^{SSout} + less_1^{SSout} \cdot N_2 + less_2^{SSout} \cdot M \rightarrow a_0^{CSin} + a_1^{CSin} \cdot N_1 + a_2^{CSin} \cdot M \geq a_0^{CSout} + a_1^{CSout} \cdot N_1 + a_2^{CSout} \cdot M$$

$$\forall \overline{X} \in \mathbb{N} : b_0^{ASin} + b_1^{ASin} \cdot N_2 + b_2^{ASin} \cdot M \geq b_0^{ASout} + b_1^{ASout} \cdot N_2 + b_2^{ASout} \cdot M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_1 + less_2^{SSin} \cdot N_2 \geq less_0^{SSout} + less_1^{SSout} \cdot N_1 + less_2^{SSout} \cdot N_2 \wedge less_0^{SSin} + less_1^{SSin} \cdot N_2 + less_2^{SSin} \cdot M \geq less_0^{SSout} + less_1^{SSout} \cdot N_2 + less_2^{SSout} \cdot M \rightarrow b_0^{CSin} + b_1^{CSin} \cdot N_2 + b_2^{CSin} \cdot M \geq b_0^{CSout} + b_1^{CSout} \cdot N_2 + b_2^{CSout} \cdot M.$$

For the second CHR rule, we have the conditions:

$$\forall \overline{X} \in \mathbb{N} : a_0^{ASin} + a_1^{ASin} \cdot N_1 + a_2^{ASin} \cdot M \geq a_0^{ASout} + a_1^{ASout} \cdot N_1 + a_2^{ASout} \cdot M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_1 + less_2^{SSin} \cdot M \geq less_0^{SSout} + less_1^{SSout} \cdot N_1 + less_2^{SSout} \cdot M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_2 + less_2^{SSin} \cdot M \geq less_0^{SSout} + less_1^{SSout} \cdot N_2 + less_2^{SSout} \cdot M \rightarrow a_0^{CSin} + a_1^{CSin} \cdot N_1 + a_2^{CSin} \cdot M \geq a_0^{CSout} + a_1^{CSout} \cdot N_1 + a_2^{CSout} \cdot M$$

$$\forall \overline{X} \in \mathbb{N} : b_0^{ASin} + b_1^{ASin} \cdot N_2 + b_2^{ASin} \cdot M \geq b_0^{ASout} + b_1^{ASout} \cdot N_2 + b_2^{ASout} \cdot M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_1 + less_2^{SSin} \cdot M \geq less_0^{SSout} + less_1^{SSout} \cdot N_1 + less_2^{SSout} \cdot M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_2 + less_2^{SSin} \cdot M \geq less_0^{SSout} + less_1^{SSout} \cdot N_2 + less_2^{SSout} \cdot M \rightarrow b_0^{CSin} + b_1^{CSin} \cdot N_2 + b_2^{CSin} \cdot M \geq b_0^{CSout} + b_1^{CSout} \cdot N_2 + b_2^{CSout} \cdot M.$$

Finally, for the CHR and Prolog predicates, to relate query sets to added sets and call sets, respectively, we have the conditions:

$$\forall \overline{X} \in \mathbb{N} : a_0^{QSin} + a_1^{QSin} \cdot X + a_2^{QSin} \cdot Y \geq a_0^{QSout} + a_1^{QSout} \cdot X + a_2^{QSout} \cdot Y \rightarrow a_0^{ASin} + a_1^{ASin} \cdot X + a_2^{ASin} \cdot Y \geq a_0^{ASout} + a_1^{ASout} \cdot X + a_2^{ASout} \cdot Y$$

$$\forall \overline{X} \in \mathbb{N} : b_0^{QSin} + b_1^{QSin} \cdot X + b_2^{QSin} \cdot Y \geq b_0^{QSout} + b_1^{QSout} \cdot X + b_2^{QSout} \cdot Y \rightarrow b_0^{ASin} + b_1^{ASin} \cdot X + b_2^{ASin} \cdot Y \geq b_0^{ASout} + b_1^{ASout} \cdot X + b_2^{ASout} \cdot Y$$

$$\forall \overline{X} \in \mathbb{N} : less_0^{QSin} + less_1^{QSin} \cdot X + less_2^{QSin} \cdot Y \geq less_0^{QSout} + less_1^{QSout} \cdot X + less_2^{QSout} \cdot Y \rightarrow less_0^{CSin} + less_1^{CSin} \cdot X + less_2^{CSin} \cdot Y \geq less_0^{CSout} + less_1^{CSout} \cdot X + less_2^{CSout} \cdot Y.$$

Note that the symbolic coefficient of query set relations have fixed values (see Example 6.1.11). Therefore, in fact, we have the conditions:

$$\forall \overline{X} \in \mathbb{N} : Y \geq 1 + X \rightarrow a_0^{ASin} + a_1^{ASin} \cdot X + a_2^{ASin} \cdot Y \geq a_0^{ASout} + a_1^{ASout} \cdot X + a_2^{ASout} \cdot Y$$

$$\forall \overline{X} \in \mathbb{N} : Y \geq 1 + X \rightarrow b_0^{ASin} + b_1^{ASin} \cdot X + b_2^{ASin} \cdot Y \geq b_0^{ASout} + b_1^{ASout} \cdot X + b_2^{ASout} \cdot Y$$

$$\forall \overline{X} \in \mathbb{N} : 0 \geq 1 \rightarrow less_0^{CSin} + less_1^{CSin} \cdot X + less_2^{CSin} \cdot Y \geq less_0^{CSout} + less_1^{CSout} \cdot X + less_2^{CSout} \cdot Y. \quad \square$$

6.2.3 Conditions for verifying rigidity

As stated earlier (see Section 6.1.2), we will only consider ground CHR(Prolog). In general, to handle non-ground programs, one can tackle the problem by applying the approach of [NDSGSK11] based on a call type analysis [JB92] resulting in constraints that are directly expressed on the symbolic coefficients of the symbolic polynomial interpretation. Then, one can add these constraints to the constraint problems for termination of ground CHR(Prolog) [NDSGSK11]. The extension to non-ground CHR(Prolog) is therefore straightforward (see Section 6.3).

6.2.4 Conditions for verifying $\mathbb{N}$-closedness

To guarantee well-foundedness of the strict partial ordering, induced by a polynomial interpretation on the call set of a CHR(Prolog) program P, we must for every predicate c/n in $Pred_P$ of P impose a condition

$$\forall \overline{X} \in \mathbb{N} : R_c^{CSs}(X_1, \ldots, X_n) \rightarrow |c(X_1, \ldots, X_n)|_{\mathbb{P}}^{s} \geq 0.$$

That is, we require that if a constraint is in its call set relation, then its size must be bounded from below by 0.

Example 6.2.6 ($\mathbb{N}$-closedness conditions)**.** *Consider the CHR(Prolog) program P of Example 6.1.2. Then, we have the following symbolic conditions for $\mathbb{N}$-closedness, one for each predicate in $Pred_P = \{a/2, b/2, less/2\}$:*

$$\forall \overline{X} \in \mathbb{N} : a_0^{CSin} + a_1^{CSin} \cdot X_1 + a_2^{CSin} \cdot X_2 \geq a_0^{CSout} + a_1^{CSout} \cdot X_1 + a_2^{CSout} \cdot X_2 \rightarrow a_0^{pos} + a_1^{pos} \cdot X_1 + a_2^{pos} \cdot X_2 - (a_0^{neg} + a_1^{neg} \cdot X_1 + a_2^{neg} \cdot X_2) \geq 0$$

$$\forall \overline{X} \in \mathbb{N} : b_0^{CSin} + b_1^{CSin} \cdot X_1 + b_2^{CSin} \cdot X_2 \geq b_0^{CSout} + b_1^{CSout} \cdot X_1 + b_2^{CSout} \cdot X_2 \rightarrow b_0^{pos} + b_1^{pos} \cdot X_1 + b_2^{pos} \cdot X_2 - (b_0^{neg} + b_1^{neg} \cdot X_1 + b_2^{neg} \cdot X_2) \geq 0$$

$$\forall \overline{X} \in \mathbb{N} : less_0^{CSin} + less_1^{CSin} \cdot X_1 + less_2^{CSin} \cdot X_2 \geq less_0^{CSout} + less_1^{CSout} \cdot X_1 + less_2^{CSout} \cdot X_2 \rightarrow less_0^{pos} + less_1^{pos} \cdot X_1 + less_2^{pos} \cdot X_2 - (less_0^{neg} + less_1^{neg} \cdot X_1 + less_2^{neg} \cdot X_2) \geq 0.$$ □

6.2.5 Conditions for verifying the decrease conditions

Decrease conditions on Prolog clauses

Symbolic decrease conditions for verifying termination of the Prolog part of a CHR(Prolog) program are formulated in the same way as such conditions are formulated in [NDSGSK11] for Prolog programs.

Thus, for each Prolog clause $(p(\bar{t}) :- p_1(\overline{t_1}), \ldots, p_m(\overline{t_m}).)$ of a CHR(Prolog) program, like in [NDSGSK11], we consider m decrease conditions:

$$\forall \overline{X} \in \mathbb{N} : R_p^{CSs}(\bar{t}) \rightarrow |p(\bar{t})|_{\mathbb{P}}^s > |p_1(\overline{t_1})|_{\mathbb{P}}^s$$

$$\forall \overline{X} \in \mathbb{N} : R_p^{CSs}(\bar{t}) \wedge R_{p_1}^{SSs}(\overline{t_1}) \rightarrow |p(\bar{t})|_{\mathbb{P}}^s > |p_2(\overline{t_2})|_{\mathbb{P}}^s$$

$$\ldots$$

$$\forall \overline{X} \in \mathbb{N} : R_p^{CSs}(\bar{t}) \wedge R_{p_1}^{SSs}(\overline{t_1}) \wedge \cdots \wedge R_{p_{m-1}}^{SSs}(\overline{t_{m-1}}) \rightarrow |p(\bar{t})|_{\mathbb{P}}^s > |p_m(\overline{t_m})|_{\mathbb{P}}^s.$$

That is, the size of the head of a Prolog clause has to be strictly greater than the size of each of the body atoms, given that each of the intermediate calls belong to their success set relations.

Example 6.2.7 (symbolic decrease conditions for Prolog clauses)**.** *Consider the CHR(Prolog) program P of Example 6.1.2. Then, for the Prolog clause of P, we have the symbolic decrease condition*

$$\forall \overline{X} \in \mathbb{N} : less_0^{CSin} + less_1^{CSin} \cdot (s_0 + s_1 \cdot X) + less_2^{CSin} \cdot (s_0 + s_1 \cdot Y) \geq less_0^{CSout} + less_1^{CSout} \cdot (s_0 + s_1 \cdot X) + less_2^{CSout} \cdot (s_0 + s_1 \cdot Y) \rightarrow less_0^{pos} + less_1^{pos} \cdot (s_0 + s_1 \cdot X) + less_2^{pos} \cdot (s_0 + s_1 \cdot Y) - (less_0^{neg} + less_1^{neg} \cdot (s_0 + s_1 \cdot X) + less_2^{neg} \cdot (s_0 + s_1 \cdot Y)) > less_0^{pos} + less_1^{pos} \cdot X + less_2^{pos} \cdot Y - (less_0^{neg} + less_1^{neg} \cdot X + less_2^{neg} \cdot Y).$$ □

Decrease conditions on propagation rules

For the decrease conditions on the CHR rules of a CHR(Prolog) program, we consider for each propagation rule

$$h_1(\overline{t_1^h}), \ldots, h_n(\overline{t_n^h}) \Rightarrow g_1(\overline{t_1^g}), \ldots, g_k(\overline{t_k^g}) \mid b_1(\overline{t_1^b}), \ldots, b_l(\overline{t_l^b}), c_1(\overline{t_1^c}), \ldots, c_m(\overline{t_m^c}).$$

of a CHR(Prolog) program $n \cdot m$ conditions of the form

$$\forall \overline{X} \in \mathbb{N} : R_{h_1}^{CSs}(\overline{t_1^h}) \wedge \cdots \wedge R_{h_n}^{CSs}(\overline{t_n^h}) \wedge R_{g_1}^{SSs}(\overline{t_1^g}) \wedge \cdots \wedge R_{g_k}^{SSs}(\overline{t_k^g}) \wedge R_{b_1}^{SSs}(\overline{t_1^b}) \wedge \cdots \wedge R_{b_l}^{SSs}(\overline{t_l^b}) \rightarrow |h_i(\overline{t_i^h})|_{\mathbb{P}}^s > |c_j(\overline{t_j^c})|_{\mathbb{P}}^s.$$

That is, for each pair of head $h_i(\overline{t_i^h})$ and body CHR constraints $c_j(\overline{t_j^c})$ of a propagation rule, there has to be a strict decrease between the size of the head constraint and the size of the body CHR constraint, given that all the head constraints belong to their call set relations and that all the guards and body built-in constraints belong to their success set relations.

Example 6.2.8 (symbolic decrease conditions for propagation rules)**.** *Consider the CHR(Prolog) program P of Example 6.1.2. Then, we have the following symbolic decrease conditions for the propagation rule of P:*

$$\forall \overline{X} \in \mathbb{N} : a_0^{CSin}+a_1^{CSin}\cdot N_1+a_2^{CSin}\cdot M \geq a_0^{CSout}+a_1^{CSout}\cdot N_1+a_2^{CSout}\cdot M \wedge b_0^{CSin}+b_1^{CSin}\cdot N_2+b_2^{CSin}\cdot M \geq b_0^{CSout}+b_1^{CSout}\cdot N_2+b_2^{CSout}\cdot M \wedge less_0^{SSin}+less_1^{SSin}\cdot N_1+less_2^{SSin}\cdot N_2 \geq less_0^{SSout}+less_1^{SSout}\cdot N_1+less_2^{SSout}\cdot N_2 \wedge less_0^{SSin}+less_1^{SSin}\cdot N_2+less_2^{SSin}\cdot M \geq less_0^{SSout}+less_1^{SSout}\cdot N_2+less_2^{SSout}\cdot M \rightarrow a_0^{pos}+a_1^{pos}\cdot N_1+a_2^{pos}\cdot M-(a_0^{neg}+a_1^{neg}\cdot N_1+a_2^{neg}\cdot M) > b_0^{pos}+b_1^{pos}\cdot(s_0+s_1\cdot N_2)+b_2^{pos}\cdot M-(b_0^{neg}+b_1^{neg}\cdot(s_0+s_1\cdot N_2)+b_2^{neg}\cdot M)$$

$$\forall \overline{X} \in \mathbb{N} : a_0^{CSin}+a_1^{CSin}\cdot N_1+a_2^{CSin}\cdot M \geq a_0^{CSout}+a_1^{CSout}\cdot N_1+a_2^{CSout}\cdot M \wedge b_0^{CSin}+b_1^{CSin}\cdot N_2+b_2^{CSin}\cdot M \geq b_0^{CSout}+b_1^{CSout}\cdot N_2+b_2^{CSout}\cdot M \wedge less_0^{SSin}+less_1^{SSin}\cdot N_1+less_2^{SSin}\cdot N_2 \geq less_0^{SSout}+less_1^{SSout}\cdot N_1+less_2^{SSout}\cdot N_2 \wedge less_0^{SSin}+less_1^{SSin}\cdot N_2+less_2^{SSin}\cdot M \geq less_0^{SSout}+less_1^{SSout}\cdot N_2+less_2^{SSout}\cdot M \rightarrow b_0^{pos}+b_1^{pos}\cdot N_2+b_2^{pos}\cdot M-(b_0^{neg}+b_1^{neg}\cdot N_2+b_2^{neg}\cdot M) > b_0^{pos}+b_1^{pos}\cdot(s_0+s_1\cdot N_2)+b_2^{pos}\cdot M-(b_0^{neg}+b_1^{neg}\cdot(s_0+s_1\cdot N_2)+b_2^{neg}\cdot M).\square$$

Decrease conditions on simpagation rules

Expressing the decrease conditions on the simpagation rules of a CHR(Prolog) program is less straightforward. These conditions verify multiset decreases. That is, for each simpagation rule

$$h_1(\overline{t_1^h}),\ldots,h_j(\overline{t_j^h}) \setminus h_{j+1}(\overline{t_{j+1}^h}),\ldots,h_n(\overline{t_n^h}) \Leftrightarrow g_1(\overline{t_1^g}),\ldots,g_k(\overline{t_k^g}) \mid b_1(\overline{t_1^b}),\ldots,b_l(\overline{t_l^b}),c_1(\overline{t_1^c}),\ldots,c_m(\overline{t_m^c}).$$

of a CHR(Prolog) program, there is a condition

$$\forall \overline{X} \in \mathbb{N} : R_{h_1}^{CSs}(\overline{t_1^h}) \wedge \cdots \wedge R_{h_n}^{CSs}(\overline{t_n^h}) \wedge R_{g_1}^{SSs}(\overline{t_1^g}) \wedge \cdots \wedge R_{g_k}^{SSs}(\overline{t_k^g}) \wedge R_{b_1}^{SSs}(\overline{t_1^b}) \wedge \cdots \wedge R_{b_l}^{SSs}(\overline{t_l^b}) \rightarrow [\![|h_{j+1}(\overline{t_{j+1}^h})|_{\mathbb{P}}^s,\ldots,|h_n(\overline{t_n^h})|_{\mathbb{P}}^s]\!] >_\mu [\![|c_1(\overline{t_1^c})|_{\mathbb{P}}^s,\ldots,|c_m(\overline{t_m^c})|_{\mathbb{P}}^s]\!].$$

That is, if all the head constraints belong to their call set relations and all the guards and body built-in constraints belong to their success set relations, then

there has to be a decrease between the multiset size of the removed head and the multiset size of the added body CHR constraints of the simpagation rule.

Furthermore, as discussed in Section 6.1.2, we will require that the strict multiset decrease $[\![|h_{j+1}(\overline{t^h_{j+1}})|^s_{\mathbb{P}}, \ldots, |h_n(\overline{t^h_n})|^s_{\mathbb{P}}]\!] >_\mu [\![|c_1(\overline{t^c_1})|^s_{\mathbb{P}}, \ldots, |c_m(\overline{t^c_m})|^s_{\mathbb{P}}]\!]$ has a singleton decreasing ranks set $[\![\rho]\!]$. Under this requirement, we additionally have for each head $h'_i(\overline{t^{h'}_i})$ of a propagation rule $(h'_1(\overline{t^{h'}_1}), \ldots, h'_{n'}(\overline{t^{h'}_{n'}}) \Rightarrow g'_1(\overline{t^{g'}_1}), \ldots, g'_{k'}(\overline{t^{g'}_{k'}}) \mid b'_1(\overline{t^{b'}_1}), \ldots, b'_{l'}(\overline{t^{b'}_{l'}}), c'_1(\overline{t^{c'}_1}), \ldots, c'_{m'}(\overline{t^{c'}_{m'}}).)$ of the program, unifiable with a body constraint $b_j(\overline{t^b_j})$ of the simpagation rule, m' conditions:

$$
\begin{array}{l}
\forall \overline{X} \in \mathbb{N} : R^{CSs}_{h_1}(\overline{t^h_1}) \wedge \cdots \wedge R^{CSs}_{h_n}(\overline{t^h_n}) \wedge \\
R^{SSs}_{g_1}(\overline{t^g_1}) \wedge \cdots \wedge R^{SSs}_{g_k}(\overline{t^g_k}) \wedge R^{SSs}_{b_1}(\overline{t^b_1}) \wedge \cdots \wedge R^{SSs}_{b_l}(\overline{t^b_l}) \wedge \\
b_j(\overline{t^b_j}) = h'_i(\overline{t^{h'}_i}) \in \mathfrak{R}^{SS}_{=/2} \wedge R^{CSs}_{h'_1}(\overline{t^{h'}_1}) \wedge \cdots \wedge R^{CSs}_{h'_{n'}}(\overline{t^{h'}_{n'}}) \wedge \\
R^{SSs}_{g'_1}(\overline{t^{g'}_1}) \wedge \cdots \wedge R^{SSs}_{g'_{k'}}(\overline{t^{g'}_{k'}}) \wedge R^{SSs}_{b'_1}(\overline{t^{b'}_1}) \wedge \cdots \wedge R^{SSs}_{b'_{l'}}(\overline{t^{b'}_{l'}}) \rightarrow |\rho|^s_{\mathbb{P}} > |c'_1(\overline{t^{c'}_1})|^s_{\mathbb{P}} \\
\ldots \\
\forall \overline{X} \in \mathbb{N} : R^{CSs}_{h_1}(\overline{t^h_1}) \wedge \cdots \wedge R^{CSs}_{h_n}(\overline{t^h_n}) \wedge \\
R^{SSs}_{g_1}(\overline{t^g_1}) \wedge \cdots \wedge R^{SSs}_{g_k}(\overline{t^g_k}) \wedge R^{SSs}_{b_1}(\overline{t^b_1}) \wedge \cdots \wedge R^{SSs}_{b_l}(\overline{t^b_l}) \wedge \\
b_j(\overline{t^b_j}) = h'_i(\overline{t^{h'}_i}) \in \mathfrak{R}^{SS}_{=/2} \wedge R^{CSs}_{h'_1}(\overline{t^{h'}_1}) \wedge \cdots \wedge R^{CSs}_{h'_{n'}}(\overline{t^{h'}_{n'}}) \wedge \\
R^{SSs}_{g'_1}(\overline{t^{g'}_1}) \wedge \cdots \wedge R^{SSs}_{g'_{k'}}(\overline{t^{g'}_{k'}}) \wedge R^{SSs}_{b'_1}(\overline{t^{b'}_1}) \wedge \cdots \wedge R^{SSs}_{b'_{l'}}(\overline{t^{b'}_{l'}}) \rightarrow |\rho|^s_{\mathbb{P}} > |c'_{m'}(\overline{t^{c'}_{m'}})|^s_{\mathbb{P}}.
\end{array}
$$

Note that success of the unification of $b_j(\overline{t^b_j})$ and $h'_i(\overline{t^{h'}_i})$ is verified using a pre-defined interargument relation for the $=/2$ built-in Prolog predicate for unification. Hence, it is not a symbolic polynomial interargument relation. Note that we define $\mathfrak{R}^{SS}_{=/2} = \{c(t_1, \ldots, t_n) = c(t'_1, \ldots, t'_n) \mid c(t_1, \ldots, t_n), c(t'_1, \ldots, t'_n) \in Atom_P \wedge \forall i : \|t_i\|_{\mathbb{P}} \approx_{\mathbb{Z}} \|t'_i\|_{\mathbb{P}}\}$.

Example 6.2.9 (symbolic decrease conditions for simpagation rules)**.** *Consider the CHR(Prolog) program P of Example 6.1.2. Then, we have the following symbolic decrease conditions for the simpagation rule of P. For the decrease condition on the simpagation rule, we have*

$$
\begin{array}{l}
\forall \overline{X} \in \mathbb{N} : a_0^{CSin} + a_1^{CSin} \cdot N_1 + a_2^{CSin} \cdot M \geq a_0^{CSout} + a_1^{CSout} \cdot N_1 + a_2^{CSout} \cdot \\
M \wedge b_0^{CSin} + b_1^{CSin} \cdot N_2 + b_2^{CSin} \cdot M \geq b_0^{CSout} + b_1^{CSout} \cdot N_2 + b_2^{CSout} \cdot \\
M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_1 + less_2^{SSin} \cdot M \geq less_0^{SSout} + less_1^{SSout} \cdot \\
N_1 + less_2^{SSout} \cdot M \wedge less_0^{SSin} + less_1^{SSin} \cdot N_2 + less_2^{SSin} \cdot M \geq less_0^{SSout} + \\
less_1^{SSout} \cdot N_2 + less_2^{SSout} \cdot M \rightarrow [\![a_0^{pos} + a_1^{pos} \cdot N_1 + a_2^{pos} \cdot M - (a_0^{neg} + a_1^{neg} \cdot \\
N_1 + a_2^{neg} \cdot M), b_0^{pos} + b_1^{pos} \cdot N_2 + b_2^{pos} \cdot M - (b_0^{neg} + b_1^{neg} \cdot N_2 + b_2^{neg} \cdot M)]\!] >_\mu \\
[\![a_0^{pos} + a_1^{pos} \cdot N_2 + a_2^{pos} \cdot M - (a_0^{neg} + a_1^{neg} \cdot N_2 + a_2^{neg} \cdot M)]\!]
\end{array}
$$

with decreasing ranks set $[\![\rho]\!]$*. For the condition on first-layer propagation for the simpagation rule, we have*

$$\forall \overline{X} \in \mathbb{N} : a_0^{CSin}+a_1^{CSin}\cdot N_1+a_2^{CSin}\cdot M \geq a_0^{CSout}+a_1^{CSout}\cdot N_1+a_2^{CSout}\cdot M \wedge b_0^{CSin}+b_1^{CSin}\cdot N_2+b_2^{CSin}\cdot M \geq b_0^{CSout}+b_1^{CSout}\cdot N_2+b_2^{CSout}\cdot M \wedge less_0^{SSin}+less_1^{SSin}\cdot N_1+less_2^{SSin}\cdot M \geq less_0^{SSout}+less_1^{SSout}\cdot N_1+less_2^{SSout}\cdot M \wedge less_0^{SSin}+less_1^{SSin}\cdot N_2+less_2^{SSin}\cdot M \geq less_0^{SSout}+less_1^{SSout}\cdot N_2+less_2^{SSout}\cdot M \wedge N_2 = N_1' \wedge M = M' \wedge a_0^{CSin}+a_1^{CSin}\cdot N_1'+a_2^{CSin}\cdot M' \geq a_0^{CSout}+a_1^{CSout}\cdot N_1'+a_2^{CSout}\cdot M' \wedge b_0^{CSin}+b_1^{CSin}\cdot N_2'+b_2^{CSin}\cdot M' \geq b_0^{CSout}+b_1^{CSout}\cdot N_2'+b_2^{CSout}\cdot M' \wedge less_0^{SSin}+less_1^{SSin}\cdot N_1'+less_2^{SSin}\cdot N_2' \geq less_0^{SSout}+less_1^{SSout}\cdot N_1'+less_2^{SSout}\cdot N_2' \wedge less_0^{SSin}+less_1^{SSin}\cdot N_2'+less_2^{SSin}\cdot M' \geq less_0^{SSout}+less_1^{SSout}\cdot N_2'+less_2^{SSout}\cdot M' \rightarrow |\rho|_{\mathbb{P}}^{s} > b_0^{pos}+b_1^{pos}\cdot(s_0+s_1\cdot N_2')+b_2^{pos}\cdot M'-(b_0^{neg}+b_1^{neg}\cdot(s_0+s_1\cdot N_2')+b_2^{neg}\cdot M').\square$$

The symbolic conditions for simpagation rules from above cannot directly be used for automating the termination proof. The reason is that the decrease condition on simpagation rules is still expressed in terms of multiset extensions and that the singleton decreasing ranks set, on the basis of which the decrease conditions for first-layer propagation are expressed, is not determined.

The generation of multiset instances

We overcome this problem by expressing the multiset decreases of the decrease conditions for simpagation rules in terms of the ordering on which the multiset extension is based. Consider for example a reduction pair $(\succeq, \succ)$ on a set $\{a, b, c\}$ with multiset extension $(\succeq_\mu, \succ_\mu)$. Then, if $[\![a, b]\!] \succ_\mu [\![c]\!]$ it can be explained by $a \succeq b \wedge a \succ c$ with decreasing ranks set $[\![a]\!]$, by $a \succ b \wedge a \approx c$ with decreasing ranks set $[\![b]\!]$ or even by $a \succ c$ with decreasing ranks set $[\![a, b]\!]$. In fact, there are many more alternative explanations. Therefore, in the RC for CHR with polynomial interpretations of Section 6.1.2, to avoid a great number of alternative explanations, we have required singleton decreasing ranks sets.

To generate the alternative explanations, also referred to as *multiset instances*, we developed an algorithm based on the following procedure.

Procedure 6.2.1 (generating multiset instances)**.** *Consider a multiset decrease* $[\![a_1, \ldots, a_n]\!] \succ_\mu [\![b_1, \ldots, b_m]\!]$*, where* $(\succeq_\mu, \succ_\mu)$ *is the multiset extension of a reduction pair* $(\succeq, \succ)$*. Furthermore, let* $n' = n - 1$ *and* $d = min(\{n', m\})$*. Then, to* generate the multiset instances *for* $[\![a_1, \ldots, a_n]\!] \succ_\mu [\![b_1, \ldots, b_m]\!]$ *in terms of* $(\succeq, \succ)$*, such that the multiset decrease has a singleton decreasing ranks set* $[\![\rho]\!]$*, we apply the following procedure:*

1. *Among the n elements of $[\![a_1, \ldots, a_n]\!]$, select an element ρ_a to represent the decreasing rank.*
2. *Among the remaining n' elements of $[\![a_1, \ldots, a_n]\!]$, select a multisubset A of $i \leq d$ elements and select for each element a in A an element b in $[\![b_1, \ldots, b_m]\!]$ such that $a \approx b$.*
3. *For the remaining $n' - i$ elements a of $[\![a_1, \ldots, a_n]\!]$: $\rho_a \succeq a$. For the remaining $m - i$ elements b of $[\![b_1, \ldots, b_n]\!]$: $\rho_a \succ b$.* □

Before we give an example, first, some discussion is in order. Consider a multiset decrease $[\![a_1, \ldots, a_n]\!] \succ_\mu [\![b_1, \ldots, b_m]\!]$ with a decreasing ranks set $[\![\rho]\!]$, where $(\succeq_\mu, \succ_\mu)$ is the multiset extension of $(\succeq, \succ)$. Then, $\rho \in [\![a_1, \ldots, a_n]\!]$. Hence, in Step 1 of Procedure 6.2.1, we select a ρ_a from $[\![a_1, \ldots, a_n]\!]$ to represent ρ. Let $[\![a_1, \ldots, a_n]\!] = [\![\rho_a]\!] \uplus [\![a_1, \ldots, a_{n'}]\!]$.

At least one of the following relations holds between the elements a_i, a_j of $[\![a_1, \ldots, a_n]\!]$: $a_i \succeq a_j$, $a_j \succeq a_i$, or a_i and a_j cannot be compared using $(\succeq, \succ)$. The a's in $[\![a_1, \ldots, a_{n'}]\!]$, for which $\rho_a \succeq a$, are considered in Step 3 of Procedure 6.2.1. The a's in $[\![a_1, \ldots, a_{n'}]\!]$, for which $a \succeq \rho_a$ or a and ρ_a cannot be compared, are considered in Step 2 of Procedure 6.2.1:

> In case a and ρ_a cannot be compared, there must exist a b in $[\![b_1, \ldots, b_m]\!]$ such that $a \approx b$. Otherwise, the decreasing ranks set, where ρ_a is assumed to be a decreasing rank, cannot be singleton. In case $a \succeq \rho_a$, there must exist a b in $[\![b_1, \ldots, b_m]\!]$ such that $a \approx b$. Otherwise, ρ_a cannot be a decreasing rank for the multiset decrease. In the special case where $\rho_a \approx a$, there can, but does not have to, exist a b such that $a \approx b$. The case where such a b exists is covered in Step 2 of Procedure 6.2.1 and the case where no such b exists is covered in Step 3 of Procedure 6.2.1.

Finally, for all body constraints that were left unconsidered, they must necessarily be strictly smaller than ρ_a to not undo the multiset decrease. This is covered by Step 3 of Procedure 6.2.1.

Thus, for $[\![a_1, \ldots, a_n]\!] \succ_\mu [\![b_1, \ldots, b_m]\!]$, where $(\succeq_\mu, \succ_\mu)$ is the multiset extension of $(\succeq, \succ)$ and where $n' = n - 1$ and $d = min(\{n', m\})$, the number of multiset instances is given by the formula:

$$n \cdot \sum_{i=0}^{d} C_i^{n'} \cdot P_i^m.$$

Here, the term n represents the selection of the decreasing rank ρ_a. The term $C_i^{n'}$ represents the selection of a multisubset of $[\![a_1, \ldots, a_n]\!]$ and P_i^m the selection of corresponding elements in $[\![b_1, \ldots, b_m]\!]$.

Example 6.2.10 (multiset instances)**.** *Consider* $[\![a, b]\!] >_\mu [\![c]\!]$*. Then, there are*

$$2 \cdot \sum_{i=0}^{1} C_i^1 \cdot P_i^1 = 4$$

multiset instances with a singleton decreasing ranks set for the multiset decrease in terms of the underlying reduction pair $(\geq, >)$*. The instances for* a *as the decreasing rank are* $a \geq b \wedge a > c$ *and* $b = c$*. The instances for* b *as the decreasing rank are* $b \geq a \wedge b > c$ *and* $a = c$*.* □

Note that when formulating the conditions for termination, we select for each of the conditions on simpagation rules some multiset instance and then formulate the conditions for first-layer propagation in terms of that multiset instance. Then, if termination cannot be proven with the selected multiset instance, we try a proof using conditions obtained by an alternative instance.

n \ m	1	2	3	4	5	6
1	1	1	1	1	1	1
2	4	6	8	10	12	14
3	9	21	39	63	93	129
4	16	52	136	292	544	916
5	25	105	365	1045	2505	5225
6	36	186	816	3006	9276	24306

Table 6.1: Number of multiset instances for increasing numbers of removed head constraints n and body CHR constraints m.

In general, for simpagation rules with many removed head and body CHR constraints, the number of alternative cases drastically increases (see Table 6.1). Therefore, in the presence of multiple simpagation rules, where for each simpagation rule a multiset instance is selected, the termination problems in terms of alternative cases can become huge. Especially this motivated the development of the modular approaches of the next chapter (see Chapter 7).

Example 6.2.11 (symbolic decrease conditions for simpagation rules w.r.t. a multiset instance)**.** *Consider the CHR(Prolog) program* P *of Example 6.1.2. Furthermore, consider the multiset decrease* $[\![a_0^{pos} + a_1^{pos} \cdot N_1 + a_2^{pos} \cdot M - (a_0^{neg} + a_1^{neg} \cdot N_1 + a_2^{neg} \cdot M), b_0^{pos} + b_1^{pos} \cdot N_2 + b_2^{pos} \cdot M - (b_0^{neg} + b_1^{neg} \cdot N_2 + b_2^{neg} \cdot M)]\!] >_\mu [\![a_0^{pos} + a_1^{pos} \cdot N_2 + a_2^{pos} \cdot M - (a_0^{neg} + a_1^{neg} \cdot N_2 + a_2^{neg} \cdot M)]\!]$ *in the symbolic decrease conditions of Example 6.2.9 on the simpagation rule. Then, considering the multiset instance* $a_0^{pos} + a_1^{pos} \cdot N_1 + a_2^{pos} \cdot M - (a_0^{neg} + a_1^{neg} \cdot N_1 + a_2^{neg} \cdot M) = a_0^{pos} + a_1^{pos} \cdot N_2 + a_2^{pos} \cdot M - (a_0^{neg} + a_1^{neg} \cdot N_2 + a_2^{neg} \cdot M)$ *with decreasing ranks*

set $[\![b_0^{pos} + b_1^{pos} \cdot N_2 + b_2^{pos} \cdot M - (b_0^{neg} + b_1^{neg} \cdot N_2 + b_2^{neg} \cdot M)]\!]$*, we have the following symbolic decrease conditions for the simpagation rule of* P*:*

$$\forall \overline{X} \in \mathbb{N} : a_0^{CSin}+a_1^{CSin}\cdot N_1+a_2^{CSin}\cdot M \geq a_0^{CSout}+a_1^{CSout}\cdot N_1+a_2^{CSout}\cdot M \wedge b_0^{CSin}+b_1^{CSin}\cdot N_2+b_2^{CSin}\cdot M \geq b_0^{CSout}+b_1^{CSout}\cdot N_2+b_2^{CSout}\cdot M \wedge less_0^{SSin}+less_1^{SSin}\cdot N_1+less_2^{SSin}\cdot M \geq less_0^{SSout}+less_1^{SSout}\cdot N_1+less_2^{SSout}\cdot M \wedge less_0^{SSin}+less_1^{SSin}\cdot N_2+less_2^{SSin}\cdot M \geq less_0^{SSout}+less_1^{SSout}\cdot N_2+less_2^{SSout}\cdot M \rightarrow a_0^{pos}+a_1^{pos}\cdot N_1+a_2^{pos}\cdot M-(a_0^{neg}+a_1^{neg}\cdot N_1+a_2^{neg}\cdot M) = a_0^{pos}+a_1^{pos}\cdot N_2+a_2^{pos}\cdot M-(a_0^{neg}+a_1^{neg}\cdot N_2+a_2^{neg}\cdot M).$$

For the decrease conditions on first-layer propagation, we have

$$\forall \overline{X} \in \mathbb{N} : a_0^{CSin}+a_1^{CSin}\cdot N_1+a_2^{CSin}\cdot M \geq a_0^{CSout}+a_1^{CSout}\cdot N_1+a_2^{CSout}\cdot M \wedge b_0^{CSin}+b_1^{CSin}\cdot N_2+b_2^{CSin}\cdot M \geq b_0^{CSout}+b_1^{CSout}\cdot N_2+b_2^{CSout}\cdot M \wedge less_0^{SSin}+less_1^{SSin}\cdot N_1+less_2^{SSin}\cdot M \geq less_0^{SSout}+less_1^{SSout}\cdot N_1+less_2^{SSout}\cdot M \wedge less_0^{SSin}+less_1^{SSin}\cdot N_2+less_2^{SSin}\cdot M \geq less_0^{SSout}+less_1^{SSout}\cdot N_2+less_2^{SSout}\cdot M \wedge N_2 = N_1' \wedge M = M' \wedge a_0^{CSin}+a_1^{CSin}\cdot N_1'+a_2^{CSin}\cdot M' \geq a_0^{CSout}+a_1^{CSout}\cdot N_1'+a_2^{CSout}\cdot M' \wedge b_0^{CSin}+b_1^{CSin}\cdot N_2'+b_2^{CSin}\cdot M' \geq b_0^{CSout}+b_1^{CSout}\cdot N_2'+b_2^{CSout}\cdot M' \wedge less_0^{SSin}+less_1^{SSin}\cdot N_1'+less_2^{SSin}\cdot N_2' \geq less_0^{SSout}+less_1^{SSout}\cdot N_1'+less_2^{SSout}\cdot N_2' \wedge less_0^{SSin}+less_1^{SSin}\cdot N_2'+less_2^{SSin}\cdot M' \geq less_0^{SSout}+less_1^{SSout}\cdot N_2'+less_2^{SSout}\cdot M' \rightarrow b_0^{pos}+b_1^{pos}\cdot N_2+b_2^{pos}\cdot M-(b_0^{neg}+b_1^{neg}\cdot N_2+b_2^{neg}\cdot M) > b_0^{pos}+b_1^{pos}\cdot(s_0+s_1\cdot N_2')+b_2^{pos}\cdot M'-(b_0^{neg}+b_1^{neg}\cdot(s_0+s_1\cdot N_2')+b_2^{neg}\cdot M').$$

Note that with a different multiset instance, we cannot prove termination. □

6.3 Towards constraints on symbolic coefficients

Our goal is to find a polynomial interpretation such that all conditions generated in the previous section are satisfied. To this end, we transform all the conditions from the previous section into Diophantine constraints on the symbolic coefficients, in the same way as this is done in [NDSGSK11].

Note that in case we verify rigidity on the basis of relevant variables, all conditions for rigidity are already Diophantine constraints, which only contain unknown coefficients but no universally quantified variables (see [NDSGSK11] and [PDS09a]). Hence, the extension to non-ground CHR is straightforward. The other conditions can all be written in the form

$$\forall \overline{X} \in \mathbb{N} : p_1 \geq q_1 \wedge \cdots \wedge p_n \geq q_n \rightarrow p_{n+1} \geq q_{n+1}, \tag{6.1}$$

where $n \geq 0$. Here all p_i, q_i are symbolic polynomials that represent polynomials in $\Pi_{\mathbb{N}}$, except for the polynomials p_{n+1} and q_{n+1} that can represent polynomials in $\Pi_{\mathbb{Z}}$ if it concerns decrease conditions. Note, however, that the symbolic coefficients and the variables of the conditions can only take values in $\mathbb{N}$ (see Definition 6.2.2). Furthermore, consider the following: if the conditions contain equalities $a = b$, these can be formulated as the conjunction of two inequalities $a \geq b \wedge b \geq a$; if the conditions contain strict inequalities $a > b$, these can be formulated as an inequality $a \geq b + 1$; and if the conclusion of the considered conditions contain more than one conjunct, these can be formulated as multiple conditions, one for each conjunct.

In the following, we recall the two-phase method from [NDSGSK11] to transform all conditions of Form 6.1 into Diophantine constraints on the unknown coefficients. In a first phase, all conditions of Form 6.1 are transformed to inequalities without premises, i.e., into constraints of the form

$$\forall \overline{X} \in \mathbb{N} : conc(p_{n+1}) - conc(q_{n+1}) \geq prem(p_1, \ldots, p_n) - prem(q_1, \ldots, q_n)$$

Here, *prem* and *conc* are two arbitrary new linear symbolic polynomials with symbolic polynomials taking values in $\mathbb{N}$. The only requirement that we have to impose is that *conc* must not be a constant. The transformation is sound: if there is a solution for the resulting constraint, then this solution also satisfies the original condition.

Thus, after this transformation, all conditions take the form

$$\forall \overline{X} \in \mathbb{N} : p \geq 0. \tag{6.2}$$

In a second phase, we transform all constraints of Form 6.2 into a set of Diophantine constraints on the unknown coefficients. The transformation is again sound: if there is a solution for the resulting Diophantine constraints, then this solution also satisfies the original constraint.

To this end, we use the straightforward transformation of [HJ98], also used in [NDSGSK11]. One only requires that all coefficients, represented as expressions on symbolic coefficients, of the polynomial p are non-negative integers.

Example 6.3.1 (from symbolic conditions to constraints on symbolic coefficients)**.** *Consider the CHR(Prolog) program P with query set I, discussed in Example 6.1.2. To prove termination of P for I automatically, we have to transform the conditions of the previous section to constraints on symbolic coefficients. In this example, we will only do so for the second condition of Example 6.2.4:*

$$\forall \overline{X} \in \mathbb{N} : less_0^{SSin} + less_1^{SSin} \cdot X + less_2^{SSin} \cdot Y \geq less_0^{SSout} + less_1^{SSout} \cdot X + less_2^{SSout} \cdot Y \rightarrow less_0^{SSin} + less_1^{SSin} \cdot (s_0 + s_1 \cdot X) + less_2^{SSin} \cdot (s_0 + s_1 \cdot Y) \geq less_0^{SSout} + less_1^{SSout} \cdot (s_0 + s_1 \cdot X) + less_2^{SSout} \cdot (s_0 + s_1 \cdot Y).$$

It is already in Form 6.1. First, we transform it to

$$\forall \overline{X} \in \mathbb{N} : conc_0 + conc_1 \cdot (less_0^{SSin} + less_1^{SSin} \cdot (s_0 + s_1 \cdot X) + less_2^{SSin} \cdot (s_0 + s_1 \cdot Y)) - (conc_0 + conc_1 \cdot (less_0^{SSout} + less_1^{SSout} \cdot (s_0 + s_1 \cdot X) + less_2^{SSout} \cdot (s_0 + s_1 \cdot Y))) \geq prem_0 + prem_1 \cdot (less_0^{SSin} + less_1^{SSin} \cdot X + less_2^{SSin} \cdot Y) - (prem_0 + prem_1 \cdot (less_0^{SSout} + less_1^{SSout} \cdot X + less_2^{SSout} \cdot Y)).$$

Note that $conc_1$ must not be 0. Then, we bring it to Form 6.2:

$$\forall \overline{X} \in \mathbb{N} : (conc_1 \cdot less_0^{SSin} + conc_1 \cdot less_1^{SSin} \cdot s_0 + conc_1 \cdot less_2^{SSin} \cdot s_0 - conc_1 \cdot less_0^{SSout} - conc_1 \cdot less_1^{SSout} \cdot s_0 - conc_1 \cdot less_2^{SSout} \cdot s_0 - prem_1 \cdot less_0^{SSin} + prem_1 \cdot less_0^{SSout}) + (conc_1 \cdot less_1^{SSin} \cdot s_1 - conc_1 \cdot less_1^{SSout} \cdot s_1 - prem_1 \cdot less_1^{SSin} + prem_1 \cdot less_1^{SSout}) \cdot X + (conc_1 \cdot less_2^{SSin} \cdot s_1 - conc_1 \cdot less_2^{SSout} \cdot s_1 - prem_1 \cdot less_2^{SSin} + prem_1 \cdot less_2^{SSout}) \cdot Y \geq 0.$$

Finally, we transform the resulting constraint to constraints on symbolic coefficients, considering that all coefficients take values in $\mathbb{N}$ and $conc_1 > 0$:

$$conc_1 \cdot less_0^{SSin} + conc_1 \cdot less_1^{SSin} \cdot s_0 + conc_1 \cdot less_2^{SSin} \cdot s_0 - conc_1 \cdot less_0^{SSout} - conc_1 \cdot less_1^{SSout} \cdot s_0 - conc_1 \cdot less_2^{SSout} \cdot s_0 - prem_1 \cdot less_0^{SSin} + prem_1 \cdot less_0^{SSout} \geq 0$$

$$conc_1 \cdot less_1^{SSin} \cdot s_1 - conc_1 \cdot less_1^{SSout} \cdot s_1 - prem_1 \cdot less_1^{SSin} + prem_1 \cdot less_1^{SSout} \geq 0$$

$$conc_1 \cdot less_2^{SSin} \cdot s_1 - conc_1 \cdot less_2^{SSout} \cdot s_1 - prem_1 \cdot less_2^{SSin} + prem_1 \cdot less_2^{SSout} \geq 0.$$ □

6.4 Solving Diophantine constraints

The previous sections showed that one can formulate all termination conditions in symbolic form and that one can transform them automatically into a set of Diophantine constraints. The problem then becomes solving a system of Diophantine constraints with the unknown coefficients as variables to prove program termination. Solving such restricted kinds of Diophantine constraints automatically has been studied intensively (see e.g., [BLNM+09], [CMTU05]

and [FGM+07]). In [FGM+07], Diophantine constraint problems are encoded as a SAT-problem, e.g., by applying the principles of unsigned arithmetic as in [CGBA+11]. It was shown in [FGM+07] that this approach is significantly more efficient than solving Diophantine constraints by dedicated solvers like [CMTU05] or by standard implementations of constraint logic programming, like in SICStus Prolog [SIC11].

Example 6.4.1 (solving Diophantine constraints)**.** *Consider the CHR(Prolog) program P with query set I, discussed in Example 6.1.2. To prove termination of P for I, combine the conditions of Example 6.2.4 for inferring the success set relations for the built-in predicates of P, the conditions of Example 6.2.5 for inferring the call set relations of the predicates of P, the conditions for $\mathbb{N}$-closedness of Example 6.2.6 and the decrease conditions of Example 6.2.7, Example 6.2.8 and Example 6.2.11 for the Prolog clauses, propagation rules and simpagation rules of P, respectively. Then, transform these conditions to a system of Diophantine constraints on symbolic coefficients as explained in Section 6.3 and finally solve the resulting system of constraints with a Diophantine constraint solver of choice. Then, one solution to the resulting system yields the polynomial interpretation of Example 6.1.4 and the inferred success, call and added set relations of Example 6.1.11, with which we have proven termination in Example 6.1.11.* □

6.5 Termination of CHR for specific queries

With our approach, we can also handle programs that only terminate for specific kinds of queries. One simple example of such a program is the CHR(Prolog) program P:

$$a(N, M) \Rightarrow less(N, M) \mid a(s(N), M).$$
$$a(N, M) \Rightarrow less(M, N) \mid a(s(N), M).$$

$$less(0, s(\_)).$$
$$less(s(X), s(Y)) :- less(X, Y).$$

If the program is queried with constraints $a(t, t)$ or with constraints $a(t_1, t_2)$ such that $P \models less(t_1, t_2)$, then the program is terminating. If queried with constraints $a(t_1, t_2)$ such that $P \models less(t_2, t_1)$ the program is not terminating. Our approach can prove termination of the program from above if it is queried with constraints $a(t, t)$ or with constraints $a(t_1, t_2)$: $P \models less(t_1, t_2)$, since we can represent this information with query set relations.

Reconsider the Merge-sort program from Example 2.1.4, of which termination has been proven in Section 2.4.2:

$$R_1 \ @ \ msort([]) \Leftrightarrow true.$$
$$R_2 \ @ \ msort([L|Ls]) \Leftrightarrow r(0, L), msort(Ls).$$
$$R_3 \ @ \ r(D, L1), r(D, L2) \Leftrightarrow less(L1, L2) \mid r(s(D), L1), a(L1, L2).$$
$$R_4 \ @ \ a(L1, L2) \setminus a(L1, L3) \Leftrightarrow less(L2, L3) \mid a(L2, L3).$$

As discussed in Section 2.4.2, the main problem in proving termination of the Merge-sort program is that R_4 may be non-terminating. However, in the intended use of the program, this will not be the case, since R_3 and R_4 can only add $a/2$ constraints of which the first argument will have a strictly smaller term-size compared to the second argument. Our approach can derive this relation for the used constraints of the $a/2$ predicate and, therefore, can use the required integer polynomials to prove termination (see Section 2.4.2).

Finally, considering the fourth rule of the Merge-sort program, it is part of a class of CHR rules that, as argued in [Frü00], are especially difficult to handle. Representative for this class is the CHR rule

$$a(X, Y), a(Y, Z) \Rightarrow a(X, Z).$$

It is a rule for graph completion, however, will not be terminating if the input graph contains cycles. E.g., if R_3 fires on $a(t_1, t_1)$ and $a(t_1, t_2)$ it adds $a(t_1, t_2)$ on which the rule is applicable in the same way.

In order to prove termination, we need to consider that the input graph is cycle-free. This property of a graph can be expressed through the existence of a strict partial ordering $\succ$ on the nodes of the graph such that for any arc $a(t_1, t_2)$ in the intended use of the program holds that $t_2 \succ t_1$. Since query set relations for constraints of the $a/2$ predicate can represent this property, termination can be proven for such specific queries to the program, like it is done in [PDS09a]. However, to handle this program automatically, we will need to consider interargument relations with a conjunction of multiple linear polynomial inequalities (see Section 9.1.3).

Chapter 7

Modularised termination proofs for CHR

Termination analysis techniques for LP often make use of dependency analysis to improve efficiency and precision of the termination analysis. In some approaches (e.g., [DDSV99]), this is done in order to detect (mutually) recursive predicates. For non-recursive predicates, no termination proof is needed, so that termination conditions are only expressed and verified for the recursive ones. Such approaches can easily be formulated in terms of CHR by means of a Strongly Connected Component (SCC) analysis on the dependency graph of a CHR program (see Section 7.3).

In other approaches (e.g., [NGSKDS08] and [SKGN10]) the termination conditions are explicitly expressed in terms of the cycles in the SCCs of the dependency graph. It was shown in [SKGN10] that such an approach is both efficient and precise. On the benchmark of the termination analysis competition [Ter10], the approach of [SKGN10] outperformed competing systems.

It turned out that there were considerable complications to achieve this in CHR, mainly because for CHR a useful notion of a "cycle" is hard to define.

7.1 Problem description

Consider for example the CHR program P:

$$R \ @ \ a, b \Leftrightarrow a.$$

For each application of R, two constraints a and b are removed, while a new constraint a is added. Since R adds constraints that are required to fire R, the application of R depends on itself. A SCCs analysis on the dependency graph of P would therefore consider R to demonstrate cyclic behaviour, while in fact, for every application of R, a constraint b is removed and never added again. Therefore, at some point in a computation of P, the b constraints will be depleted and thus P will terminate.

A more accurate analysis of cyclic behaviour in CHR could therefore still rely on a SCCs analysis, but could also verify for each SCC whether there exists a dependency for each head of the rules of a component, provided for by the component itself. Unfortunately, such an approach is still not very accurate. Consider for example the CHR program P:

$$R \ @ \ a, a \Leftrightarrow a.$$

For R, which is part of a SCC, both heads are provided for from within the component itself. However, for every application of R, two constraints a are removed while only one constraint a is added. Repeated application of R therefore results in depletion of these a constraints and thus P terminates.

Therefore, in order for a SCC in a CHR program to demonstrate cyclic behaviour, we must guarantee that the component is *self-sustainable*. That is, when the rules of a SCC are applied, they must provide the constraints to fire the same rules again. In contrast to CHR, the SCCs of a Prolog program are self-sustainable by default. This is because Prolog is a single-headed programming language. Therefore, each cycle in the dependency graph of a Prolog program corresponds to rule applications that have the potential to be repeated.

In CHR, to characterise the notion of a self-sustainable SCC, we need to identify the SCCs of a CHR program for which there exist *multisets* of rule applications that can be repeated. Consider for example the CHR program P:

$$R_1 \ @ \ a, a \Leftrightarrow b, b. \qquad R_2 \ @ \ b \Leftrightarrow a.$$

As can be verified it is non-terminating. That is, when R_1 is applied once, it adds two b constraints, allowing R_2 to be applied twice. By applying R_2 twice, the input required for application of R_1 is provided for, thus resulting in a loop.

In this chapter, we introduce the notion of *self-sustainability* for a set of CHR rules. It provides a rather precise approximation of cyclic behaviour in CHR. We characterise self-sustainability by a system of linear inequalities. Each solution to this system identifies a multiset, based on a set of CHR rules, such that if the multiset is *traversed* —i.e., all rules in the multiset are applied once— it provides all required constraints to potentially traverse the multiset again.

Unfortunately, we cannot use this concept directly as a basis for termination analysis similar to [NGSKDS08] and [SKGN10] for LP. The reason is that for any SCC in a CHR program that demonstrates cyclic behaviour, the system of linear inequalities for self-sustainability has infinitely many solutions, yielding an infinite set of minimal configuration that have the potential to be repeated. In [PDS10], we describe a rather inefficient approach, using Hilbert bases [Sch86], to cope with this problem.

Alternatively, we can use the concept of self-sustainability in a "negative" way. If we can show that a SCC in a CHR program is *not* self-sustainable —i.e., the system of linear inequalities for self-sustainability has no solution—, then this proves that the rules of that SCC cannot be part of an infinite computation. Therefore, we can ignore this SCC in the termination analysis (see Section 7.3). By combining this idea with the approach of the previous chapter, it turns out that both precision and efficiency of the analysis is improved (see Chapter 8).

Our approach will be restricted to the abstract CHR semantics. Hence, we discuss our approach in terms of simplification only (see Section 2.2). This is not a problem since if a SCC of a CHR program is not self-sustainable for the abstract semantics, it cannot be self-sustainable for any more refined semantics. Therefore, the approach of this section is also applicable to the theoretical CHR semantics, however, will consider SCCs with propagation self-sustainable by default. Note that in Section 9.2, we provide intuitions regarding self-sustainability of propagation under the theoretical CHR semantics.

7.2 The CHR dependency graph and CHR net

In CHR, the head constraints of a rule represent the constraints *required* for rule application. The body CHR constraints represent the constraints *newly available* after rule application. To represent these sets, we introduce *abstract CHR constraints*.

First, we introduce the alternative notation

$$R_i \ @ \ H^i_1, \ldots, H^i_{n_i} \Leftrightarrow G^i_1, \ldots, G^i_{k_i} \mid B^i_1, \ldots, B^i_{l_i}, C^i_1, \ldots, C^i_{m_i}.$$

for a simplification rule, as we need to be able to identify the rule to which CHR and built-in constraints belong. Note that in the following, by G^i, we mean the conjunction $G^i_1 \wedge \cdots \wedge G^i_{k_i}$ of the guards of a simplification rule R_i.

Definition 7.2.1 (abstract CHR constraint)**.** *An* abstract CHR constraint *is a pair $\mathcal{C} = (C, B)$, where C is a CHR constraint and B a conjunction of built-in constraints. The denotation of an abstract CHR constraint is given by a mapping $\wp : (C, B) \mapsto \{C\sigma \mid \exists\theta : CT \models B\sigma\theta\}$.*

In a CHR program P, we distinguish between two types of abstract CHR constraints. An abstract input constraint *$in^i_j = (H^i_j, G^i)$ represents the CHR constraints that can be used to match the head H^i_j of a CHR rule R_i, without invalidating the conjunction G^i of guards of R_i. An* abstract output constraint *$out^i_j = (C^i_j, G^i)$ represents the CHR constraints that become newly available after rule application and are related to the body CHR constraints C^i_j of R_i.* □

By $\mathcal{I}n_{R_i}$ and $\mathcal{O}ut_{R_i}$, we represent, respectively, the sets of abstract input and output constraints of a rule R_i. By $\mathcal{I}n_P$ and $\mathcal{O}ut_P$, we represent, respectively, the abstract input and output constraints of a CHR program P.

Example 7.2.1 (abstract CHR constraints of Primes)**.** *Consider a variant P of the Primes program of Example 2.1.4:*

$$R_1 \ @ \ primes(2) \Leftrightarrow prime(2).$$
$$R_2 \ @ \ primes(N) \Leftrightarrow N > 2 \mid Np \ is \ N - 1, prime(N), primes(Np).$$
$$R_3 \ @ \ prime(M), prime(N) \Leftrightarrow div(M, N) \mid prime(M).$$

We can derive the following abstract input and output constraints for P:

- $\mathcal{I}n_P = \mathcal{I}n_{R_1} \cup \mathcal{I}n_{R_2} \cup \mathcal{I}n_{R_3} = \{in^1_1, in^2_1, in^3_1, in^3_2\}$, *where*
  $in^1_1 = (primes(2), true)$ $\quad in^2_1 = (primes(N), N > 2)$
  $in^3_1 = (prime(M), div(M, N))$ $\quad in^3_2 = (prime(N), div(M, N))$
- $\mathcal{O}ut_P = \mathcal{O}ut_{R_1} \cup \mathcal{O}ut_{R_2} \cup \mathcal{O}ut_{R_3} = \{out^1_1, out^2_1, out^2_2, out^3_1\}$, *where*
  $out^1_1 = (prime(2), true)$ $\quad out^2_1 = (prime(N), N > 2)$
  $out^2_2 = (primes(Np), N > 2)$ $\quad out^3_1 = (prime(M), div(M, N))$ □

The rules of a CHR program P relate abstract inputs to abstract outputs. We call this relation the *rule transition relation* of P.

Definition 7.2.2 (rule transition relation)**.** *A rule transition of an abstract CHR program P is an ordered pair $T_i = (\mathcal{I}n_{R_i}, \mathcal{O}ut_{R_i})$, relating the set of abstract input constraints $\mathcal{I}n_{R_i} = \{in^i_1, \ldots, in^i_{n_i}\}$ of $R_i \in P$ to the set*

of abstract output constraints $\mathcal{O}ut_{R_i} = \{out^i_1, \ldots, out^i_{m_i}\}$ *of* R_i*. The rule transition relation* $\mathcal{T}_P = \{T_i \mid R_i \in P\}$ *of* P *is the set of rule transitions of* P*.* □

Example 7.2.2 (rule transition relation of Primes)**.** *The Primes program* P *from Example 7.2.1 defines three rule transitions:*

- $\mathcal{T} = \{T_1, T_2, T_3\}$*, where*
  $T_1 = (\{in^1_1\}, \{out^1_1\})$ $\quad T_2 = (\{in^2_1\}, \{out^2_1, out^2_2\})$
  $T_3 = (\{in^3_1, in^3_2\}, \{out^3_1\})$

□

Abstract output constraints relate to abstract input constraints by a *match transition relation*. This second kind of relation is the result of a dependency analysis between abstract constraints, relating the constraints newly available after rule application to constraints required for rule application.

Definition 7.2.3 (match transition relation)**.** *A match transition of an abstract CHR program* P *is an ordered pair* $M_{(i,j,k,l)} = (out^i_j, in^k_l)$*, relating an output* $out^i_j = (C^i_j, G^i)$ *of* $\mathcal{O}ut_P$ *to an input* $in^k_l = (C^k_l, G^k)$ *of* $\mathcal{I}n_P$ *such that* $\exists\theta : CT \models (C^i_j = C^k_l \wedge G^i \wedge G^k)\theta$*. The match transition relation* $\mathcal{M}_P$ *is the set of all match transitions* $M_{(i,j,k,l)}$ *in* P*.* □

Note that a match transition exists for an abstract output and input if the intersection of their denotation is non-empty, i.e., $\wp(out^i_j) \cap \wp(in^k_l) \neq \emptyset$.

Example 7.2.3 (match transition relation of Primes)**.** *The Primes program* P *from Example 7.2.1 defines the match transition relation*

$$\mathcal{M}_P = \{M_{(1,1,3,1)}, M_{(1,1,3,2)}, M_{(2,1,3,1)}, M_{(2,1,3,2)}$$
$$M_{(2,2,1,1)}, M_{(2,2,2,1)}, M_{(3,1,3,1)}, M_{(3,1,3,2)}\}.$$

□

As in LP, the dependency graph of a CHR program is a directed graph, where the nodes represent rules and the directed arcs dependencies between rules.

Definition 7.2.4 (CHR dependency graph)**.** *A* CHR dependency graph $\mathcal{D}_P$ *of an abstract CHR program* P *is an ordered tuple* $\langle T, D\rangle$ *of nodes* T*, one for each transition in the rule transition relation* $\mathcal{T}_P$ *of* P*, and directed arcs* D*, one for each ordered pair of transitions for which a match transition, connecting an output of the first element of the pair to an input of the second element of the pair, exists in the match transition relation* $\mathcal{M}_P$ *of* P*.* □

For a SCCs analysis, a CHR dependency graph is sufficient, however, for self-sustainability we rely in the next sections on a CHR net instead.

Definition 7.2.5 (CHR net)**.** *A* CHR net $\mathcal{N}_P$ *of an abstract CHR program* P *is a quadruple* $\langle \mathcal{I}n_P, \mathcal{O}ut_P, \mathcal{T}_P, \mathcal{M}_P \rangle$*. Here, respectively,* $\mathcal{I}n_P$ *and* $\mathcal{O}ut_P$ *are the abstract input and output constraints of* P *and* $\mathcal{T}_P$ *and* $\mathcal{M}_P$ *the rule and match transition relations of* P*.* □

Notice that a CHR net corresponds to a bipartite hypergraph. We illustrate both notions in Figure 7.1 for the Primes program of Example 7.2.1.

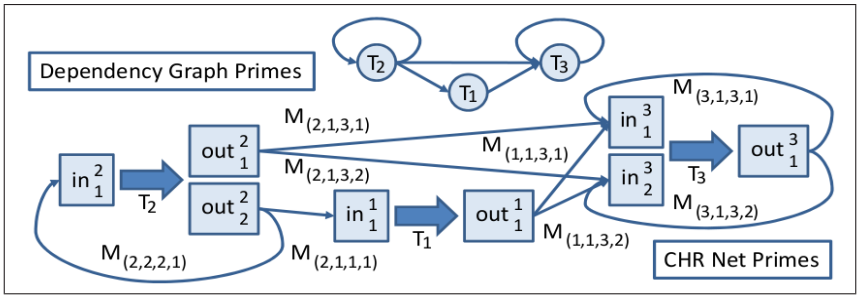

Figure 7.1: Dependency graph and CHR net for Primes

7.3 Self-sustainable SCCs of a CHR program

To derive the SCCs of a CHR program, i.e., the sets of CHR rules of a CHR program that are strongly connected, we perform a SCCs analysis on the dependency graph of the CHR program. This is similar to such an analysis in LP and can be done efficiently using Tarjan's algorithm [Tar72].

Example 7.3.1 (SCCs of Primes)**.** *Based on the dependency graph,* $\mathcal{D}_P = (\{T_1, T_2, T_3\}, \{(T_1, T_3), (T_2, T_1), (T_2, T_2), (T_2, T_3), (T_3, T_3)\})$*, of Primes of Figure 7.1, we derive two SCCs,* $\{T_2\}$ *and* $\{T_3\}$*.* □

We have the following property of SCCs w.r.t. program termination.

Theorem 7.3.1. *Let* P *be a CHR program and let* C *be a SCC of* P*. If* C *terminates for every query and if* $P \setminus C$ *terminates for every query, then* P *terminates for every query.* □

Proof. By definition of a SCC, if each SCC in the dependency graph of P is contracted to a single node, the resulting graph G is a directed acyclic graph. Therefore, if $P \setminus C$ is terminating for any query, its rules add a finite number of

constraints using constraints from the initial query. Since C is terminating for any query, by using the constraints added by the rules of $P \setminus C$ and those from the initial query, the rules of C can only add a finite number of constraints. Then, any rule of $P \setminus C$ applicable on the constraints produced by C can only be applied a finite number of times as $P \setminus C$ is terminating for any query. As furthermore these rules of $P \setminus C$ do not produce constraints required by C considering acyclicity of G, P must be terminating for any query. □

Note that Theorem 7.3.1 can easily be adapted in terms of an intended use I for P. Thus, a termination proof for each of the SCCs of a CHR program, yields a proof of termination on the entire program.

7.3.1 Self-sustainable SCCs

Consider the non-terminating example program from Section 7.1:

$$R_1 \ @ \ a, a \Leftrightarrow b, b. \qquad R_2 \ @ \ b \Leftrightarrow a.$$

Its dependency graph and CHR net are shown in Figure 7.2. As can be verified, in Figure 7.2, $in_1^1 = in_2^1 = out_1^2 = (a, true)$ and $out_1^1 = out_2^1 = in_1^2 = (b, true)$. For this example program, there is only one SCC: $\{T_1, T_2\}$.

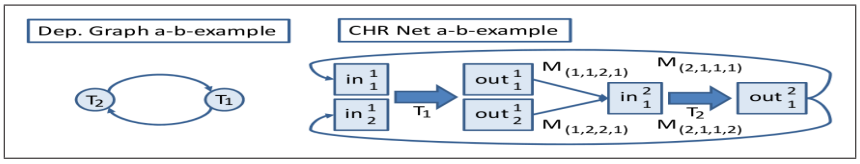

Figure 7.2: Dependency graph and CHR net for the *a*-*b*-example

What we want to characterise, in the notion of self-sustainability for such a SCC, is whether we can duplicate some of the transitions of this SCC, such that for the resulting multiset of transitions and its *extended CHR net*, it is so that a multisubset of all $M_{(i,j,k,l)}$ match transitions exists, that maps a multisubset of all the out_q^p nodes *onto all of* the in_s^r nodes.

Returning to our example, consider the multiset $[\![T_1, T_2, T_2]\!]$ based on $\{T_1, T_2\}$. In Figure 7.3, we expand the CHR net of Figure 7.2 to represent all match transitions for this multiset.

Clearly, from the multiset of 8 match transitions in Figure 7.3, we can select 4 transitions, for example $[\![M_{(1,1,2,1)}, M_{(1,2,2,1)}, M_{(2,1,1,1)}, M_{(2,1,1,2)}]\!]$, such that

these transitions define a function from a multisubset of all out_q^p nodes *onto* all in_s^r nodes. This function is represented in Figure 7.3 in the thick arrows.

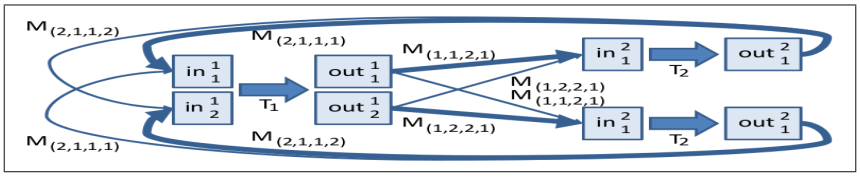

Figure 7.3: Expanded CHR net for $[\![T_1, T_2, T_2]\!]$

Note that in this example, the multisubsets of out_q^p nodes on which the function is defined is equal to the entire multiset $[\![out_1^1, out_2^1, out_1^2, out_1^2]\!]$ of out_q^p nodes. In general, this is not necessary: out_q^p nodes are allowed to be "unused" by in_s^r nodes. It suffices that a CHR constraint is produced for each in_s^r node.

We generalise the construction in the example above in the following definition. In this definition, we start off from a set of CHR rules C and its CHR net $\mathcal{N}_C = \langle \mathcal{I}n_C, \mathcal{O}ut_C, \mathcal{T}_C, \mathcal{M}_C \rangle$. In the definition we will consider a multiset, with universe the rule transitions $\mathcal{T}_C$, and will denote it by $\mu_{\mathcal{T}}$. Note that given such a multiset $\mu_{\mathcal{T}}$, there are associated multisets $\mu_{\mathcal{I}n}$ and $\mu_{\mathcal{O}ut}$, with universes $\mathcal{I}n_C$ and $\mathcal{O}ut_C$ respectively. If a transition $T_i \in \mathcal{T}_C$ occurs n_i times in $\mu_{\mathcal{T}}$, then all its in_j^i and out_l^i abstract constraints occur n_i times in $\mu_{\mathcal{I}n}$ and $\mu_{\mathcal{O}ut}$, respectively.

Given $\mu_{\mathcal{T}}$, there is also an associated multiset $\mu_{\mathcal{M}}$, with universe $\mathcal{M}_C$. If transitions T_i and T_k occur respectively n_i and n_k times in $\mu_{\mathcal{T}}$, then all $M(i,j,k,l) \in \mathcal{M}_C$ occur $n_i \cdot n_k$ times in $\mu_{\mathcal{M}}$.

Definition 7.3.1 (self-sustainability of a set of CHR rules)**.** *Let C be a set of simplification rules and $\mathcal{N}_C = \langle \mathcal{I}n_C, \mathcal{O}ut_C, \mathcal{T}_C, \mathcal{M}_C \rangle$ the CHR net of C. The set C is self-sustainable iff there exist*

- *a non-empty multiset $\mu_{\mathcal{T}}$, with associated multisets $\mu_{\mathcal{I}n}$, $\mu_{\mathcal{O}ut}$ and $\mu_{\mathcal{M}}$,*
- *a multiset $\mu'_{\mathcal{O}ut} \sqsubseteq \mu_{\mathcal{O}ut}$, and*
- *a multiset $\mu'_{\mathcal{M}} \sqsubseteq \mu_{\mathcal{M}}$,*

such that $\mu'_{\mathcal{M}}$ defines a function from $\mu'_{\mathcal{O}ut}$ onto *$\mu_{\mathcal{I}n}$.* □

Before we illustrate this concept on our running example Primes, we provide an alternative numeric characterisation.

Proposition 7.3.1 (numeric characterisation of self-sustainability)**.** *Let C be a set of simplification rules and $\mathcal{N}_C = \langle \mathcal{I}n_C, \mathcal{O}ut_C, \mathcal{T}_C, \mathcal{M}_C \rangle$ its CHR net. The set C is self-sustainable iff there exist multisets $\mu_{\mathcal{T}}$, with associated multisets $\mu_{\mathcal{I}n}$, $\mu_{\mathcal{O}ut}$ and $\mu_{\mathcal{M}}$, $\mu'_{\mathcal{O}ut} \sqsubseteq \mu_{\mathcal{O}ut}$ and $\mu'_{\mathcal{M}} \sqsubseteq \mu_{\mathcal{M}}$, such that $\sum_{i:T_i \in \mathcal{T}_C} \mu_{\mathcal{T}}(T_i) \geq 1$ and such that*

$$\forall out^i_j \in \mathcal{O}ut_C : \sum_{k,l:M_{(i,j,k,l)} \in \mathcal{M}_C} \mu'_{\mathcal{M}}(M_{(i,j,k,l)}) \leq \mu_{\mathcal{T}}(T_i) \text{ and}$$

$$\forall in^i_j \in \mathcal{I}n_C : \sum_{k,l:M_{(k,l,i,j)} \in \mathcal{M}_C} \mu'_{\mathcal{M}}(M_{(k,l,i,j)}) = \mu_{\mathcal{T}}(T_i). \qquad \square$$

Proof. The first inequality expresses that $\mu_{\mathcal{T}}$ is non-empty. The second kind of inequalities —one for each out^i_j in $\mathcal{O}ut_C$— state that for an abstract output out^i_j, the number of match transitions $M_{(i,j,k,l)}$ in $\mu'_{\mathcal{M}}$ that have (i,j) as their first two arguments does not exceed the number of occurrences of T_i in $\mu_{\mathcal{T}}$. This corresponds to stating that $\mu'_{\mathcal{M}}$ can be regarded as a function defined on a multisubset $\mu'_{\mathcal{O}ut}$ of $\mu_{\mathcal{O}ut}$.

The third kind of inequalities —one for each in^i_j in $\mathcal{I}n_C$— state that for an abstract input constraint in^i_j, the number of match transitions $M_{(k,l,i,j)}$ in $\mu'_{\mathcal{M}}$ that have (i,j) as their last two arguments is exactly the number of occurrences of T_i in $\mu_{\mathcal{T}}$. This means that the function defined by $\mu'_{\mathcal{M}}$ is onto $\mu_{\mathcal{I}n}$. $\square$

Example 7.3.2 (self-sustainable SCCs for Primes)**.** *Consider the SCCs $C_1 = \{T_2\}$ and $C_2 = \{T_3\}$, as given in Example 7.3.1, of the Primes program P of Example 7.2.1. While the first component of the program is self-sustainable, the second is not. That is, at some point in a computation of C_2, the prime/1 constraints to fire the rule will be depleted. Consider their CHR nets*

$$\mathcal{N}_{C_1} = \langle \{in^2_1\}, \{out^2_1, out^2_2\}, \{T_2\}, \{M_{(2,2,2,1)}\} \rangle \quad \text{and}$$
$$\mathcal{N}_{C_2} = \langle \{in^3_1, in^3_2\}, \{out^3_1\}, \{T_3\}, \{M_{(3,1,3,1)}, M_{(3,1,3,2)}\} \rangle.$$

Let us denote $\mu'_{\mathcal{M}}(M_{(i,j,k,l)})$ as $m_{(i,j,k,l)}$ and $\mu_{\mathcal{T}}(T_i)$ as t_i, where all $m_{(i,j,k,l)}$ and t_i are natural numbers. Then, we can characterise self-sustainability of C_1 and C_2, respectively, by the systems of linear inequalities

$$t_2 \geq 1 \qquad m_{(2,2,2,1)} \leq t_2 \qquad m_{(2,2,2,1)} = t_2 \qquad \text{and}$$
$$t_3 \geq 1 \qquad m_{(3,1,3,1)} + m_{(3,1,3,2)} \leq t_3 \qquad m_{(3,1,3,1)} = t_3 \qquad m_{(3,1,3,2)} = t_3.$$

If a solution to these systems exists, the underlying component is self-sustainable. C_1 is clearly self-sustainable, while C_2 is not. $\square$

Several comments with respect to Definition 7.3.1 and Proposition 7.3.1 are in order. First, the statement that $\mu'_{\mathcal{M}}$ defines a function from $\mu'_{\mathcal{O}ut}$ onto $\mu_{\mathcal{I}n}$ is imprecise. In fact, $\mu'_{\mathcal{M}}$ defines a set of functions from $\mu'_{\mathcal{O}ut}$ onto $\mu_{\mathcal{I}n}$. This is because $\mu'_{\mathcal{O}ut}$ and $\mu_{\mathcal{I}n}$ are multisets and not sets.

Therefore, if in^i_j or out^k_l occur multiple times in $\mu_{\mathcal{I}n}$, respectively $\mu'_{\mathcal{O}ut}$, it is unclear which mapping is defined by an element $M_{(k,l,i,j)}$ of $\mu'_{\mathcal{M}}$. Looking back at the *a*-*b*-example of Figure 7.3, there are four different functions in this graph, all corresponding to the multiset $\mu'_{\mathcal{M}} = [\![M_{(1,1,2,1)}, M_{(1,2,2,1)}, M_{(2,1,1,1)}, M_{(2,1,1,2)}]\!]$. The thick lines in the figure represent one of these, but selecting other arcs, with the same labels, produces the other three.

Apart from this, for a fixed multiset $\mu_{\mathcal{T}}$, there can be several mappings $\mu'_{\mathcal{M}}$ that map a multiset $\mu'_{\mathcal{O}ut}$ *onto* $\mu_{\mathcal{I}n}$. This is because there can be different abstract constraints out^k_l in $\mu_{\mathcal{O}ut}$ that all have a match transition to some abstract constraint in^i_j and this may give alternative candidates for $\mu'_{\mathcal{M}}$.

Finally, by considering different multisets $\mu_{\mathcal{T}}$ based on $\mathcal{T}_C$, we may obtain a very large number of solutions for $\mu'_{\mathcal{M}}$ in Definition 7.3.1. In the context of the *a*-*b*-example, consider a multiset $\mu_{\mathcal{T}} = [\![T_1, T_1, T_2, T_2, T_2, T_2]\!]$. From the fact that there are twice as many T_2 rules than T_1 rules, it should be intuitively clear that it again allows to find multisets $\mu'_{\mathcal{O}ut}$ and $\mu'_{\mathcal{M}} : \mu'_{\mathcal{O}ut} \to \mu_{\mathcal{I}n}$, with the latter being *onto*. Because of the increased multiplicity of the rule transitions, the number of different functions that $\mu'_{\mathcal{M}}$ represents in this context is much higher than for the previous $\mu_{\mathcal{T}}$. Moreover, it turns out that by increasing the multiplicity of the rule transitions, we can construct concrete functions associated to $\mu'_{\mathcal{M}}$ that cannot be obtained as the union of multiple concrete functions associated to a solution for a $\mu_{\mathcal{T}}$ with lower multiplicity of rule transitions (see [PDS09c]).

This observation implies that we are unable to use the notion of a self-sustainable set of rules as a direct basis for a termination analysis. Such an approach would have to identify a finite set of minimal self-sustainable cycles and then prove that all these are terminating. But since in CHR there are in general an infinite set of minimal cycles, such an approach is not feasible.

In [PDS10], to tackle the problem of an infinite number of minimal cycles, a new concept of minimality is introduced. This concept relies on a finite constructive set of solutions for the inequalities representing self-sustainability, called the Hilbert basis [Sch86]. This approach is slow, with little gain in precision.

Fortunately, there is another way for using the notion of a self-sustaining set of rules. After determining the SCCs of a CHR program, we can verify which SCCs are not self-sustainable and disregard such SCCs. This observation is based on the following theorem.

Theorem 7.3.2. *If a CHR program P is not self-sustainable, then P terminates for every query.* □

To prove Theorem 7.3.2, we apply Ramsey's theorem [Ram30].

Theorem 7.3.3 (Ramsey's theorem)**.** *Let $\mathcal{A} = \{\langle a, b\rangle \mid a, b \in \mathbb{N} \wedge a < b\}$, $\mathcal{L}$ be a finite set of colours and let $\mathcal{F} : \mathcal{A} \to \mathcal{L}$ be a mapping associating the elements of $\mathcal{A}$ with colours from $\mathcal{L}$. Then, there is a colour $f \in \mathcal{L}$ and an infinite set $X \subseteq \mathbb{N}$ such that $\mathcal{F}(\langle a, b\rangle) = f$ for each $a, b \in X$ for which $a < b$.* □

Note that the application of Ramsey's theorem to termination analysis is not new, see e.g., [BCG+07]. Next, we give the proof of Theorem 7.3.2.

Proof. Let Σ_P be the set of CHR states for a CHR program P. Let $\mathcal{I}n_P$ be the set of abstract input constraints of P. Then, we define a mapping, $|.|_{in_i} : \Sigma_P \to \mathbb{N}$, mapping CHR states to the number of constraints in the state that are also in $\wp(in_i)$, the denotation of the abstract input constraint $in_i \in \mathcal{I}n_P$. Thus, $|.|_{in_i}$ represents the number of constraints in a CHR state that can potentially be used to match the head of a rule, represented by in_i.

By contradiction, we assume that P is non-terminating. Therefore, there must exist an infinite computation

$$Q_0 \to_P Q_1 \to_P \cdots \to_P Q_n \to_P \ldots$$

of P on some initial CHR state Q_0.

If P is not self-sustainable, then for any two states Q and Q' in the computation such that Q' is obtained from Q (by a multiset of rule applications), we have by Definition 7.3.1 that $\exists in_i \in \mathcal{I}n_P : |Q|_{in_i} > |Q'|_{in_i}$. That is, at least one head of a CHR rule of P must exist for which the available constraints decrease between Q and Q'. Note that each $|.|_{in_i}$ induces a well-founded ordering $>_i$ on Σ_P. Therefore, if P is not self-sustainable, we have that

$$\forall k \geq 0 : \forall j > k, \exists in_i \in \mathcal{I}n_P : Q_k >_i Q_j,$$

where j and k are subscripts of the states in the considered computation.

Applying Ramsey's theorem, we consider a set of ordered tuples $\mathcal{A} = \{\langle a, b\rangle \mid a, b \in \mathbb{N} \wedge a < b\}$ and a finite set of colours $\mathcal{L} = \{>_i \mid in_i \in \mathcal{I}n_P\}$. Furthermore, we define a mapping $\mathcal{F} : \mathcal{A} \to \mathcal{L}$, as follows. For any $\langle a, b\rangle \in \mathcal{A}$, we map $\langle a, b\rangle$ to some $>_i$ of $\mathcal{L}$ for which $Q_a >_i Q_b$ holds. Since P is not self-sustainable, there must exist at least one such a $>_i \in \mathcal{L}$ for any $\langle a, b\rangle \in \mathcal{A}$.

Therefore, by Ramsey's theorem, there exists a $>_i \in \mathcal{L}$ and an infinite subset $X \in \mathbb{N}$ such that for any $a, b \in X$ if $a > b$, $\mathcal{F}(\langle a, b\rangle) = >_i$ holds. This yields an

infinite subsequence of CHR states in the considered infinite computation for which a strict decrease on $>_i$ exists. Since this contradicts the well-foundedness of $>_i$, no infinite computation of P can exist. Thus, by Lemma 2.3.1, P terminates for every query. □

7.3.2 Non-self-sustainable SCCs

From here on, we refer to self-sustainable as '*selfs*' and use the matrix form: $A \times X \leq B$; to represent systems of linear inequalities.

Example 7.3.3 (system in matrix form for Primes)**.** *We revisit Example 7.3.2 and represent for C_1 and C_2 their systems of inequalities in matrix form. For C_1, we have*

$$\begin{bmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 0 \end{bmatrix} \times \begin{bmatrix} t_2 \\ m_{(2,2,2,1)} \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}.$$

For C_2, we have

$$\begin{bmatrix} -1 & 1 & 0 \\ 1 & -1 & 0 \\ -1 & 0 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 1 \\ -1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} t_3 \\ m_{(3,1,3,1)} \\ m_{(3,1,3,2)} \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}.$$

Note that the equalities of the original system are replaced by two inequalities. E.g., $m_{(2,2,2,1)} = t_2$ is replaced by $m_{(2,2,2,1)} \geq t_2$ and $m_{(2,2,2,1)} \leq t_2$. □

In order to prove that a component is *not selfs*, we need to prove that no positive integer solution exists for the variables of the linear inequalities representing that it is *selfs*. Thus, we need to prove that such a system is *infeasible*.

Definition 7.3.2 (feasible)**.** *A system, $S = A \times X \leq B$, is* (in)feasible *iff S has (not) a solution in $\mathbb{R}^+$.* □

The following lemma is due to Farkas [Far02].

Lemma 7.3.1 (Farkas' lemma)**.** *Let $S = A \times X \leq B$ be a system of linear inequalities. Then, S is feasible iff $\forall P \geq 0 : A^T \times P \geq 0 \rightarrow B^T \times P \geq 0$. Alternatively, S is infeasible iff $\exists P \geq 0 : A^T \times P \geq 0 \wedge B^T \times P < 0$.* □

Note that Farkas' lemma is only applicable to the real case: if a real matrix P exists such that $P \geq 0 \wedge A^T \times P \geq 0 \wedge B^T \times P < 0$, then the original system S is infeasible. Therefore, it has no solution in $\mathbb{N} \subset \mathbb{R}^+$ and must be *non-selfs*.

Infeasibility has received much attention in linear programming (see e.g., [Bea96]) and several approaches exist to tackle the problem. It is not in our intention to improve on these approaches. However, to evaluate our approach (see Chapter 8), we formulate a simple test, where we represent infeasibility as a constraint problem on symbolic coefficients. That is, we introduce a symbolic matrix P', for each infeasibility problem, of the same dimensions as P, with symbolic coefficients p_i, one for each position in the matrix. Then, we derive constraints on the symbolic coefficients, based on $P' \geq 0 \wedge A^T \times P' \geq 0 \wedge B^T \times P' < 0$.

Example 7.3.4 (infeasibility for Primes)**.** *We revisit Example 7.3.3 and formulate infeasibility of the systems of linear inequalities. That is, let P_1 for C_1 be a (4×1)-matrix of (integer) symbolic coefficients $p_0^1, p_1^1, \ldots, p_3^1$, all greater or equal to* 0*. Then, to prove* non-selfs *for C_1, we need to satisfy:*

$$\begin{bmatrix} -1 & 1 & -1 & -1 \\ 1 & -1 & 1 & 0 \end{bmatrix} \times P_1 \geq 0 \quad \textit{and} \quad \begin{bmatrix} 0 & 0 & 0 & -1 \end{bmatrix} \times P_1 < 0.$$

We derive the following problem, where $\forall i \in \{0, 1, \ldots, 3\} : p_i^1 \geq 0$ and

$$-p_0^1 + p_1^1 - p_2^1 - p_3^1 \geq 0 \qquad p_0^1 - p_1^1 + p_2^1 \geq 0 \qquad -p_3^1 < 0.$$

There can be no solution to this constraint problem, thus C_1 cannot be proven non-selfs*. For C_2, we have*

$$\begin{bmatrix} -1 & 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 \end{bmatrix} \times P_2 \geq 0 \quad \textit{and} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \times P_2 < 0.$$

We derive the following problem, where $\forall i \in \{0, 1, \ldots, 5\} : p_i^2 \geq 0$ and

$$\begin{array}{ll} -p_0^2 + p_1^2 - p_2^2 + p_3^2 - p_4^2 - p_5^2 \geq 0 & p_0^2 - p_1^2 + p_4^2 \geq 0 \\ p_2^2 - p_3^2 + p_4^2 \geq 0 & -p_5^2 < 0. \end{array}$$

One solution is: $p_0^2 = 0$, $p_1^2 = 1$, $p_2^2 = 0$, $p_3^2 = 1$, $p_4^2 = 1$ and $p_5^2 = 1$. Therefore, C_2 is non-selfs *and thus must be terminating.* □

Note that there are programs on which the approach of Chapter 6, based on Chapter 4, fails, but on which the non-selfs test of this chapter succeeds. One example of such a program is $\{(R_1 \ @ \ a, a \Leftrightarrow b.), (R_2 \ @ \ b \Leftrightarrow a.)\}$ (see Section

5.2.2). That is, to prove termination with the RC for CHR, for R_1, the size of a must be greater or equal to b. For R_2, a has to be strictly smaller than b.

The non-self-sustainability test of this chapter is able to handle such problems and integrated with the approach of the previous chapter improves both the efficiency and the precision of the termination analysis (see Chapter 8). Note, however, that also the transformational approach of [PSDS07a, PSDS07b] and the additive approach of [Frü00] can be used to handle such programs.

Therefore, the main motivation of this chapter is to provide for a simple and efficient test so that the SCCs in CHR programs that are not self-sustainable (and thus do not demonstrate cyclic behaviour) can be detected. These can then be ignored when a proof of termination is attempted using the RC for CHR with polynomial interpretations (see Chapter 8), as such providing an additional approach for proving part of a CHR program terminating.

Chapter 8

T*CoP: Termination of CHR on top of Prolog

To evaluate the practicality of the approaches of Chapter 6 and Chapter 7, we developed *T*CoP*, a **T**ermination analyser for **C**HR **o**n top of **P**rolog (see Figure 8.1), available for download at [T*C10].

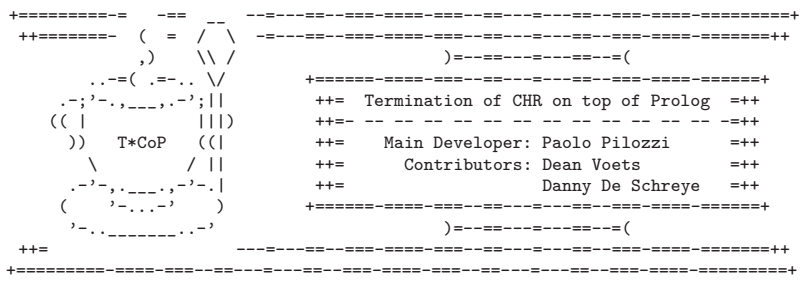

Figure 8.1: T*CoP Header

In this chapter, we give first an overview of T*CoP's implementation. Afterwards, we provide an evaluation with T*CoP on a benchmark of practical and constructed programs.

8.1 Implementation

T*CoP is a termination analyser for CHR on top of Prolog, implemented in SWI-Prolog [SP10]. T*CoP is based on the RC for CHR with polynomial interpretations from Chapter 6, where the search for polynomial interpretations, like in [NDSGSK11], is automated using a constraint-based approach (see Section 6.2). T*CoP also implements the non-self-sustainability test from Chapter 7, based on a SCCs analysis.

The mode of operation of T*CoP is illustrated in Figure 8.2.

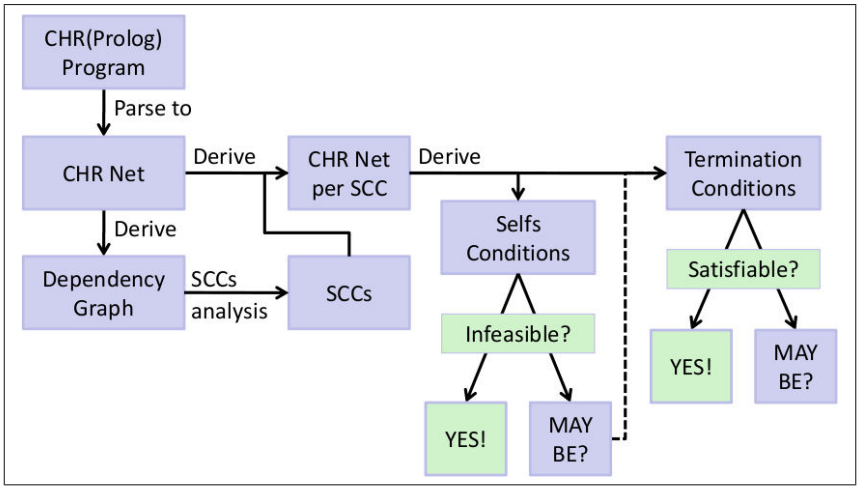

Figure 8.2: Mode of operation of T*CoP

The input of T*CoP is a CHR(Prolog) program with a module declaration. The module declaration defines the exported constraints of the program, where an empty export is considered empty. Given a CHR(Prolog) program, T*CoP derives, as its internal representation of the CHR(Prolog) program, a CHR net (see Section 7.2), that also includes the Prolog part of the program. In doing so, T*CoP removes the answer constraints from the program, i.e., the CHR constraints in the body of CHR rules that cannot be used to fire any of the CHR rules of the program. As such a great number of alternatives for termination, due to multiset instances, may be avoided (see Section 6.2.5). Furthermore, T*CoP performs a reachability analysis, based on the exported constraints of the program, such that only those parts of the program that can be reached by external calls are considered. Therefore, an empty export results in an empty CHR net and thus a trivially terminating program.

Then, based on the CHR net, T*CoP derives a dependency graph, on which it performs a SCCs analysis using Tarjan's algorithm [Tar72]. Finally, using the result of the SCCs analysis, CHR nets are constructed for each of the SCCs.

8.1.1 The non-self-sustainability test

For each CHR net of the SCCs of the program, T*CoP formulates self-sustainability of the SCC as a system of linear constraints with positive integer solutions (see Section 7.3). In this phase of the analysis, T*CoP applies a number of optimisations w.r.t. self-sustainability of a SCC. That is, if it concerns a SCC of the Prolog part of the program or a SCC that contains propagation, then T*CoP considers that SCC self-sustainable by default. If the SCC only consist of simpagation rules, then T*CoP first verifies whether some heads are not provided for from within the SCC (see Section 7.1) and considers those SCCs non-self-sustainable by default. For the remaining SCCs, T*CoP verifies infeasibility of the system of linear constraints for self-sustainability.

8.1.2 Verifying the RC for CHR

The SCCs that are possibly self-sustainable are verified against the RC for CHR with polynomial interpretations (see Chapter 6). First, for each of the these SCCs, T*CoP formulates call set conditions, $\mathbb{N}$-closedness conditions and decrease conditions (see Section 6.2). Then, T*CoP determines the rules that, considering the exported constraints, can directly or indirectly make calls to the SCC. For these rules, T*CoP includes call set conditions as well. Furthermore, based on the exported constraints, also query set conditions are included. Then, T*CoP determines the rules on which the SCC depends and on which the rules that can make calls to the SCC depend. For these rules, T*CoP includes success set conditions. Finally, T*CoP transforms the symbolic conditions for termination of the SCC to constraints on symbolic coefficients (see Section 6.4) and verifies satisfaction of these constraints to prove termination.

Recall that the RC for CHR with polynomial interpretations, in the presence of multi-headed simpagation rules, yields a set of disjunctive constraint problems (see Section 6.2.5). If one of these constraint problems can be satisfied, we prove termination. In generating the disjunctive constraint problems, T*CoP first verifies satisfiability for the alternatives corresponding to the RC for CHR with propagation of Chapter 3. Since in [VDSP08] it was demonstrated that the RC for CHR with propagation is applicable to many programs in practice, this approach is usually more efficient in a sequential approach.

Currently, T*CoP does not allow for the use of higher-order symbolic polynomials in termination proofs. However, T*CoP has been designed to make it easy to extend on this. That is, the application of a symbolic polynomial interpretation to termination conditions has been defined in a separate SWI-Prolog module. In that module, definitions for symbolic level mappings, norms and interargument relations can easily be adapted to the higher-order case, such that these can be used in the termination proofs instead.

8.1.3 The Diophantine constraint solver

Finally, to solve the constraints resulting from infeasibility of self-sustainability or from the RC for CHR with polynomial interpretations, we use a finite domain solver in $\mathbb{N}$. Note that, in the case of the test for non-self-sustainability, this yields an incomplete method since Farkas' Lemma is expressed in $\mathbb{R}^+$ (see Section 7.3.2). In practice, however, a solver in $\mathbb{N}$ seems sufficient. That is, not a single non-self-sustainable SCC could be constructed, that required a solver in $\mathbb{R}^+$ to be proven non-self-sustainable.

The finite domain solver of T*CoP reformulates a system of constraints as a SAT problem, which then is solved by MiniSAT2 [Min10]. To reformulate a system of constraints as a SAT-problem, T*CoP implements a transformation based on *unsigned integer arithmetic* (see e.g., [CGBA+11]). In such a transformation, to obtain a finite representation of the constraint problem, we restrict on the number of bits for representing positive integer numbers. E.g., a 2-bit representation allows for positive integers in the interval $[0, 3]$ and a 3-bit representation for positive integers in the interval $[0, 7]$.

Finally, note that since it was not in our intention to make a state-of-the-art termination analysis tool for CHR(Prolog), but a proof-of-concept analyser to demonstrate the practicality of the approaches of the previous chapters, we did not experiment with other SAT solvers, dedicated solvers or solvers that implement an infeasibility test.

8.2 Evaluation

In this section, we evaluate T*CoP on a benchmark of 87 CHR(Prolog) programs, available for download at [T*C10].

8.2.1 Interpreting the results

We evaluated T*CoP using a Linux system (Ubuntu 10.04 - Kernel: 2.6.32-32-generic) with CPU: Intel(R) Pentium(R) D CPU 2.80GHz; and Memory: 2048160 kB. The results of this analysis are given in Tables 8.1, 8.2, 8.3 and 8.4, and are organised as follows.

Each Table is subdivided into four columns. The first column is a reference to the programs for which we have analysed termination using T*CoP. This column is subdivided into two columns, the left of which gives the name of the program and the right of which gives the number of SCCs in the program.

The second column is related to the test for non-self-sustainability of a SCC. This column is subdivided in 3 pairs of columns. The first pair refers to a non-self-sustainability test performed by a transformation to SAT with 1-bit representations, thus using the positive integer domain $[0, 1]$ for the variables of the Diophantine constraint problems. In the left column of the pair we denote whether the SCC can be proven terminating in this setting, using '**?**' for "*the SCC is maybe terminating*" and '**Y**' for "*the SCC is terminating*". In the right column of the pair we give the time for analysis in seconds. The second and third pair of columns within the column for the non-self-sustainability test are organised in the same way, considering 2-bit and 3-bit representations.

CHR(Prolog)		Non-selfs test						RC for CHR with $\mathbb{P}$				Total
Program	♯	1	sec	2	sec	3	sec	1	sec	2	sec	sec
ackermann	1	?	0,141	?	0,411	?	0,849	?	1,224	?	4,398	7,030
average	1	?	0,099	?	0,300	?	0,636	?	1,528	Y	6,111	8,679
binlog	1	?	0,018	?	0,059	?	0,125	?	0,069	Y	1,334	1,610
booland	0											0,006
boolcard	1	?	0,101	?	0,295	?	0,608	Y	2,159			3,170
concat	1	?	0,083	?	0,261	?	0,662	?	8,076	?	37,01	46,09
convert	1	?	0,096	?	0,289	?	0,600	Y	1,650			2,640
dfsearch	1	?	0,097	?	0,294	?	0,601	Y	1,709			2,706
diff	1	?	0,018	?	0,057	?	0,125	Y	0,766			0,966
factorial	1	?	0,019	?	0,058	?	0,129	Y	1,039			1,250
fib	1	?	0,001	?	0,001	?	0,001	Y	1,632			1,641
gcd1	1	?	0,035	?	0,183	?	0,713	Y	0,575			1,511
gcd2	1	?	0,040	?	0,209	?	0,768	?	0,606	?	2,213	3,842
genint1	1	?	0,001	?	0,001	?	0,001	Y	0,632			0,639
genint2	1	?	0,001	?	0,001	?	0,001	Y	0,615			0,622
genint3	1	?	0,070	?	0,218	?	0,594	Y	2,016			2,902
joinlists	1	?	0,054	?	0,175	?	0,407	Y	1,153			1,794
max	1	Y	0,001					Y	0,215			0,004
mean	1	Y	0,090					Y	6,127			0,095
modulo	1	?	0,018	?	0,059	?	0,125	Y	0,725			0,931
oddeven	1	?	0,021	?	0,060	?	0,126	?	0,065	Y	1,290	1,566
primes1	1	?	0,001	?	0,001	?	0,001	?	0,068	Y	1,601	1,672
revlist	1	?	0,018	?	0,059	?	0,125	Y	0,710			0,917
som	1	Y	0,001					Y	1,274			0,005
succadd	1	?	1,484	?	4,965	?	10,857	Y	5,007			22,32
toyama	0											0,003
weight	1	?	0,021	?	0,060	?	0,124	Y	0,840			1,049
zebra	1	?	0,207	?	0,635	?	1,680	Y	1,705			4,233
ztoa	1	?	0,019	?	0,058	?	0,127	?	0,181	?	0,526	0,916

Table 8.1: Results of T*CoP on practical programs with one SCC.

The third column is related to proofs of termination based on the RC for CHR with polynomial interpretations. This column is subdivided in two pairs of columns with a similar meaning as the columns for the non-self-sustainability test, i.e., one for 1-bit representations and the other for 2-bit representations.

Finally, the fourth column provides the total time for analysis in seconds and should be interpreted as follows. The non-self-sustainability test is performed before we try to prove termination using the RC for CHR with polynomial interpretations. Both approaches are performed incrementally. That is, if a proof of termination with 1-bit representations cannot be found, a proof of termination is tried with 2-bit representations, and so on.

To avoid confusion, note that in the tables, even though a termination proof is obtained on the basis of the non-self-sustainability test, we will always include the results of verifying the same SCC with the RC for CHR with polynomial interpretations. The reason for this is to be able to interpret the effect of the non-self-sustainability test on the total time for analysis.

Furthermore, note that since all non-self-sustainable SCCs can be proven non-self-sustainable with at most a 3-bit representation ($[0, 7]$) we have limited the incrementation at 3-bit representations. Since all self-sustainable SCCs in the benchmark, that can be handled by T*CoP using the RC for CHR with polynomial interpretations, can be proven terminating with at most a 2-bit representation ($[0, 3]$), we have limited the incrementation of bit representations for the RC for CHR with polynomial interpretations at 2.

CHR(Prolog)		Non-selfs test						RC for CHR with $\mathbb{P}$				Total
Program	♯	1	sec	2	sec	3	sec	1	sec	2	sec	sec
even	1	Y	0,001					Y	0,178			0,551
	2	?	0,001	?	0,001	?	0,001	Y	0,543			
genalg1	1	?	0,001	?	0,001	?	0,001	Y	0,797			2,276
	2	?	0,001	?	0,001	?	0,001	Y	1,468			
genalg2	1	?	0,001	?	0,001	?	0,001	Y	0,754			56,18
	2	?	0,001	?	0,001	?	0,001	?	17,88	Y	37,53	
genalg3	1	?	0,001	?	0,001	?	0,001	Y	0,772			13,48
	2	?	0,001	?	0,001	?	0,001	Y	12,69			
genalg4	1	?	0,001	?	0,001	?	0,001	Y	0,695			183,6
	2	?	0,001	?	0,001	?	0,001	?	16,18	?	166,8	
genalg5	1	?	0,001	?	0,001	?	0,001	Y	0,734			101,5
	2	?	0,001	?	0,001	?	0,001	?	16,61	Y	84,17	
mergesort1	1	?	0,058	?	0,172	?	0,339	Y	1,721			2,314
	2	Y	0,019					Y	0,937			
mergesort2	1	?	0,059	?	0,170	?	0,342	Y	1,928			7,426
	2	Y	0,020					Y	1,126			
	3	?	0,025	?	0,067	?	0,143	Y	3,524			
	4	?	0,001	?	0,001	?	0,001	Y	1,145			
permutations	1	?	0,058	?	0,167	?	0,351	Y	1,873			12,76
	2	?	0,001	?	0,001	?	0,001	?	1,120	?	7,920	
	3	Y	0,018					?	2,181	?	15,42	
	4	Y	0,001					Y	0,775			
	5	?	0,019	?	0,058	?	0,129	Y	1,039			
primes2	1	?	0,022	?	0,059	?	0,125	Y	0,634			0,846
	2	Y	0,001					Y	0,318			
primes3	1	?	0,028	?	0,074	?	0,167	?	0,039	Y	0,978	1,331
	2	Y	0,039					?	0,522	Y	0,253	

Table 8.2: Results of T*CoP on practical programs with multiple SCCs.

On the other hand, in a termination analysis competition, like for LP [Ter10], we would not limit on bit representations but on total time for analysis, where the general setting is 120 seconds. Since the competition [Ter10] is run on a multi-core system, we would run the non-self-sustainability test in parallel with the verification of the RC for CHR and would, depending on available cores, also analyse alternatives due to bit representations or due to multiset instances concurrently. Therefore, the results of this section in terms of total time for analysis, when seen in the setting of a competition, can be much improved.

Finally, note that the timings do not add up to the total time for analysis. This is because we did not include timings for parsing and for performing a SCCs analysis. These timings, since they are not necessary for the evaluation, were excluded due to space restrictions. Note however that for all programs of the benchmark, they were all in the range of $0,003$ to $0,008$ seconds.

8.2.2 Evaluating the results

We evaluated T*CoP on a benchmark of 40 practical programs (see Tables 8.1 and 8.2) of which 11 programs consist of multiple SCCs (see Table 8.2). All of these programs are terminating programs with the exception of $genalg4$:

$$
\begin{array}{l}
:- \mathit{module}(\mathit{genalg}, [a/2, b/2]).\\
:- \mathit{use\_module}(\mathit{library}(\mathit{chr})).\\
:- \mathit{chr\_constraint}\ a/2, b/2.
\end{array}
$$

$$
\begin{array}{l}
a(N1, M), b(N2, M) \Rightarrow \mathit{less}(N1, M), \mathit{less}(N2, M) \mid b(s(N2), M).\\
a(N1, M), b(N2, M) \Leftrightarrow \mathit{less}(N1, M), \mathit{less}(N2, M) \mid a(N2, M).
\end{array}
$$

$$
\begin{array}{l}
\mathit{less}(0, s(\_)).\\
\mathit{less}(s(X), s(Y)) :- \mathit{less}(X, Y).
\end{array}
$$

Note that the program is almost identical to the program of Example 6.1.2, with the exception of a single programming error resulting in non-termination of the program. It is included in the benchmark to demonstrate the ease with which programming errors can be made in CHR and, therefore, leave the debugging of the program as an exercise to the reader.

With T*CoP, of the 39 terminating programs, we were able to prove termination of 34. Of the 55 terminating SCCs in the benchmark, we were able to prove 50 terminating. Note that the benchmark includes all of the practical programs discussed in the previous chapters. In Table 8.1, these are: $gcd1$, the Greatest Common Divisor program of Example 1.3.1 and Section 2.4.1; and fib, the Fibonacci program of Sections 3.3.3 and 4.4.3. In Table 8.2, these

are: *mergesort*2, the Merge-sort program of Example 2.1.4 and Sections 2.4.2, 3.3.1, 4.4.1 and 6.5; *primes*2, the Primes program of Sections 3.3.2 and 4.4.2; *genalg*3, the Genetic Algorithm of Example 6.1.2, used as a running example in Chapter 6; and *primes*3, the Primes program of Example 7.2.1, used as a running example in Chapter 7.

We, furthermore, evaluated T*CoP on a benchmark of 45 constructed programs, possibly consisting of more than one SCC (see Tables 8.3 and 8.4). The programs of Table 8.3 are included to demonstrate the effect of an increasing number of heads and bodies in simpagation rules. Each program corresponds to a different configuration that requires a different multiset instance to be proven terminating. We have named these programs $constrHhBbN$, where H represents the number of removed head constraints, B the number of body constraints and N some configuration. The programs of Table 8.4 are the constructed programs that are discussed in the previous chapters or are variants thereof. That is: $constrdiff08$ and $constrdiff09$ are the programs of Section 3.3.4, with $constrdiff01, \ldots, constrdiff07$ related programs; $constrbinc01$ is the bounded increase problem of Example 6.1.1, with $constrbinc02$ and $constrbinc03$ related programs; $constrprop01, \ldots, constrprop04$ are some of the non-self-sustainable example programs of Section 7.1 and some more difficult non-self-sustainable programs; $constrnont01, \ldots, constrnont06$ are non-terminating programs of varying complexity to verify the correct behaviour of T*CoP; and finally, $constrspeq01$ is the first program of Section 6.5, $constrspeq02$ the program for graph completion of Section 6.5 and $constrspeq03$ a variant of $constrspeq02$.

CHR(Prolog)		Non-selfs test						RC for CHR with $\mathbb{P}$				Total
Program	♯	1	sec	2	sec	3	sec	1	sec	2	sec	sec
constr1h1b01	1	?	0,019	?	0,058	?	0,125	Y	0,281			0,487
constr1h2b01	1	?	0,036	?	0,107	?	0,222	Y	0,437			0,806
constr1h2b02	1	?	0,038	?	0,105	?	0,225	Y	0,781			1,153
constr2h1b01	1	Y	0,048					Y	0,690			0,052
constr2h1b02	1	Y	0,048					Y	1,190			0,051
constr2h1b03	1	Y	0,047					Y	1,012			0,050
constr2h1b04	1	Y	0,049					Y	0,520			0,052
constr2h1b05	1	Y	0,048					Y	1,572			0,052
constr2h1b06	1	Y	0,048					Y	0,806			0,052
constr2h1b07	1	Y	0,048					Y	0,946			0,052
constr2h2b01	1	?	0,080	?	0,265	?	0,664	Y	0,802			1,815
constr2h2b02	1	?	0,082	?	0,265	?	0,681	Y	1,024			2,056
constr2h2b03	1	?	0,082	?	0,262	?	0,666	Y	3,759			4,773
constr2h2b04	1	?	0,082	?	0,265	?	0,672	Y	1,283			2,306
constr2h2b05	1	?	0,081	?	0,263	?	0,671	Y	1,160			2,179
constr2h2b06	1	?	0,083	?	0,261	?	0,666	Y	1,333			2,347
constr2h2b07	1	?	0,083	?	0,263	?	0,672	Y	1,472			2,494
constr2h2b08	1	?	0,084	?	0,267	?	0,671	Y	5,058			6,084
constr2h2b09	1	?	0,084	?	0,263	?	0,667	Y	1,690			2,707
constr2h2b10	1	?	0,082	?	0,264	?	0,672	?	8,048	Y	6,999	16,07
constr2h2b11	1	?	0,085	?	0,265	?	0,668	?	9,067	Y	9,844	19,93
constr2h2b12	1	?	0,082	?	0,262	?	0,672	?	10,05	Y	11,70	22,77

Table 8.3: Results of T*CoP on constructed programs that require different multiset instances for a proof of termination.

As can be verified in Tables 8.1, 8.2, 8.3 and 8.4, in general, the test for non-self-sustainability, when it is able to prove termination of a SCC, results in a significant reduction in time for analysis compared to the RC for CHR with polynomial interpretations. Otherwise, if the non-self-sustainability test cannot prove termination, the overhead is relatively small. Note, however, that there is always the exception, as can be verified for the *succadd* program of Table 8.1. Furthermore, recall that we apply a number of optimisations regarding the test for non-self-sustainability, as discussed in the Implementation section of this chapter. In case such an optimisation is applied, the time for analysis is always under $0,001$ seconds, however, was rounded up to $0,001$ seconds.

Furthermore, as can be verified in Table 8.1 and Table 8.2, there are not that many practical programs with non-self-sustainable SCCs that were considered. Note, however, that in case the programs did contain non-self-sustainable SCCs, we were always able to prove non-self-sustainability using 1-bit representations. In Table 8.4, therefore, we included two programs that require 2-bit and 3-bit representations, constructed for this purpose. For most practical programs, it seems therefore sufficient to limit the test at 2-bit representations.

CHR(Prolog)		Non-selfs test						RC for CHR with $\mathbb{P}$				Total
Program	♯	1	sec	2	sec	3	sec	1	sec	2	sec	sec
constrbinc01	1	?	0,001	?	0,001	?	0,001	Y	0,538			1,669
	2	?	0,019	?	0,059	?	0,123	Y	0,922			
constrbinc02	1	?	0,018	?	0,057	?	0,125	Y	0,558			0,762
constrbinc03	1	?	0,001	?	0,001	?	0,001	Y	0,540			20,54
	2	?	0,022	?	0,059	?	0,126	?	1,267	?	18,51	
constrdiff01	1	?	0,001	?	0,001	?	0,001	?	0,990	Y	3,740	4,736
constrdiff02	1	?	0,001	?	0,001	?	0,001	Y	6,446			6,454
constrdiff03	1	?	0,001	?	0,001	?	0,001	Y	10,82			10,83
constrdiff04	1	?	0,001	?	0,001	?	0,001	Y	9,395			9,403
constrdiff05	1	?	0,001	?	0,001	?	0,001	?	16,42	Y	55,06	71,48
constrdiff06	1	?	0,001	?	0,001	?	0,001	?	11,06	Y	32,77	43,83
constrdiff07	1	?	0,120	?	0,487	?	1,041	Y	3,867			52,90
	2	?	0,001	?	0,001	?	0,001	?	12,02	Y	34,57	
	3	?	0,001	?	0,001	?	0,001	Y	0,791			
constrdiff08	1	?	0,461	?	1,390	?	3,041	Y	22,00			26,90
constrdiff09	1	?	0,001	?	0,001	?	0,001	?	112,8	Y	394,5	507,4
constrnont01	1	?	0,001	?	0,001	?	0,001	Y	0,645			33,61
	2	?	0,096	?	0,307	?	0,593	?	2,453	?	29,51	
constrnont02	1	?	0,068	?	0,202	?	0,435	?	0,189	?	0,786	1,685
constrnont03	1	?	0,098	?	0,295	?	0,653	?	0,239	?	1,004	2,294
constrnont04	1	?	0,103	?	0,349	?	1,314	?	42,05	?	174,3	218,1
constrnont05	1	?	0,054	?	0,178	?	0,413	?	3,571	?	13,55	17,77
constrnont06	1	?	0,158	?	0,459	?	0,997	?	2,027	?	8,324	11,97
constrprop01	1	Y	0,047					Y	0,478			0,050
constrprop02	1	?	0,111	Y	0,345			?	1,070	?	4,207	0,460
constrprop03	1	Y	0,130					Y	2,905			0,133
constrprop04	1	?	0,323	?	1,054	Y	2,219	?	12,26	?	74,99	3,602
constrspeq01	1	?	0,001	?	0,001	?	0,001	Y	0,732			16,11
	2	?	0,096	?	0,292	?	0,615	?	2,484	Y	11,88	
constrspeq02	1	?	0,028	?	0,076	?	0,161	Y	1,713			12,32
	2	?	0,001	?	0,001	?	0,001	?	1,522	?	7,873	
	3	?	0,001	?	0,001	?	0,001	Y	0,941			
constrspeq03	1	?	0,001	?	0,001	?	0,001	Y	4,380			5,261
	2	?	0,001	?	0,001	?	0,001	?	0,876	Y	11,88	

Table 8.4: Results of T*CoP on various constructed programs.

There are still many CHR(Prolog) programs that T*CoP cannot handle. We included 7 programs, representative for important classes of programs that currently cannot be handled:

- One important program that we cannot handle with T*CoP is the *ackermann* program of Table 8.1, representative for a class of programs consisting of non-primitive recursion. To handle such programs, it is required to consider a lexicographical ordering on the constraints of the program (see Section 9.1.4).
- A second class of programs that cannot be handled by T*CoP are programs where termination depends on CHR constraints used as intermediate calls (see Section 5.2.1 and Section 9.1.1). Examples of such programs are the *ztoa* program and the *gcd*2 program of Table 8.1.
- A third class of programs that cannot be handled by T*CoP are programs for which termination requires a multiplicative extension (see Section 5.2.2 and Section 9.1.1). An example of such a program is the *concat* program of Table 8.1.
- A fourth class of programs that cannot be handled by T*CoP are programs for which interargument relations need to consider a conjunction of multiple polynomial inequalities to prove termination (see Section 9.1.3). Examples of such programs are the (fourth SCC of the) *permutations* program of Table 8.2 and the *constrspeq*02 program of Table 8.4.
- A fifth class of programs that cannot be handled by T*CoP are programs that need an integer polynomial interpretation of functor symbols. An example of such a program is the *constrbinc*03 program of Table 8.4. It is a variant of *constrbinc*01, where the bounded increase problem is specified within a sub-term (see Section 9.1.2).

Note that there is only one terminating program that T*CoP can handle and that cannot be proven terminating within a 120 seconds time limit. This is $constrdiff09$ of Table 8.4, the second program of Section 3.3.4. The reason for this is that a proof of termination, using the RC for CHR with polynomial interpretations, will fail on the first 16 multiset instances generated. Also note that *constrspeq*03 of Table 8.4, although it is a variant of *constrspeq*02 of Table 8.4, can be proven terminating. The reason is that for *constrspeq*03 we do not need a conjunction of polynomial inequalities to prove termination.

Finally, as can be verified in Table 8.3, on average, the time for analysis increases with the number of removed head and body constraints (see Table 6.1). This is mainly due to the increased complexity of the conditions. The variations in time for analysis for different configurations of a rule with the same number of head and body constraints are more drastic and are related to a requirement for a different multiset instance, of which the order of generation is fixed in advance, to prove termination (see Section 8.1.2).

Chapter 9

Practice: Conclusions

In the second part of the thesis, we discussed the automation of termination proofs for CHR(Prolog). First, in Chapter 6, we developed termination conditions for CHR(Prolog) that, in contrast to the termination conditions of the first part of the thesis, can be verified in a finite number of steps. These conditions are based on integer polynomial interpretations, extending the approach of [PDS09a] (and [NDSGSK11]) based on positive integer polynomial interpretations. As such, we are also able to handle bounded increase problems and programs that only terminate for specific kinds of queries.

To automate the verification process, we proceeded in light of the constraint-based approach of [DDSV99] and [NDSGSK11]. That is, we formulated the conditions for termination using symbolic polynomials and symbolic polynomial interargument relations. Then, we transformed the resulting symbolic conditions to constraints on the symbolic coefficients of these conditions. This transformation involved formulating multiset decreases in terms of the underlying ordering of a multiset extension, yielding alternative constraint problems that each, when satisfiable, imply termination of a CHR program.

In Chapter 7, we modularised termination proofs for CHR, scaling down the termination problem for CHR. There it was shown that it is sufficient to consider only the SCCs of a CHR(Prolog) program for proving termination of the program. Furthermore, in Chapter 7, we introduced the notion of a CHR cycle, defined as a self-sustainable SCC of a CHR program. Based on this characterisation of CHR cycles, we introduced an approach for proving non-self-sustainability of the SCCs of a CHR program, providing us with an

additional way of proving part of a CHR program terminating.

Finally, in Chapter 8, we introduced T*CoP, an automated termination analyser for CHR(Prolog) implementing the approaches of Chapters 6 and 7. Furthermore, we provided an evaluation of T*CoP on a benchmark of practical and constructed examples.

This chapter concludes the discussion on automating termination proofs for CHR. In the remainder of this chapter, we will discuss a number of limitations of the approaches of Chapters 6 and 7 and provide intuitions to overcome these limitations. As such, we outline future work regarding improved frameworks for automated termination analysis of CHR.

9.1 Limitations of the RC for CHR with $\mathbb{P}$

There are a number of limitations to the RC for CHR with polynomial interpretations of Chapter 6 that are important to resolve.

9.1.1 Limitations inherited from the RC for CHR

A first limitation to the approach of Chapter 6, related to the approach of Chapter 4, is that we cannot handle *success set relations for CHR predicates*. As discussed in Section 5.2.1, this is not straightforward to overcome.

A second limitation is due to *the requirement of different extensions* as discussed in Section 5.2.2. Many of these problems can be handled with the approach of Chapter 7, the additive approach of [Frü00] or the transformational approach of [PSDS07a, PSDS07b].

A third limitation is related to the condition on first-layer propagation of the RC for CHR of Chapter 4 (see Section 5.2.3), which in *the presence of single-headed propagation rules* can be too restrictive.

9.1.2 Interpreting functor symbols in $\Pi_{\mathbb{Z}}$

As can be verified in Definition 6.1.2, polynomial interpretations cannot interpret terms in $\Pi_{\mathbb{Z}}$. The underlying reason for this was that polynomial interargument relations of Definition 6.1.5 cannot represent relations at the level of sub-terms and therefore that the interpretation of functor symbols would need to be in $\Pi_{\mathbb{N}}$ anyway. Therefore, if a programmer specifies a bounded

increase problem at the level of functor symbols, the program cannot be proven terminating. An example of such a program is considered in the Evaluation section of Chapter 8 and can be characterised by the following CHR rule:

$$a(sub(X,Y)) \Leftrightarrow less(X,Y) \mid a(sub(s(X),Y)).$$

Although there are solutions to this problem involving more expressive interargument relations, it seems preferable, on the basis of the rule from above, to flatten the program instead. As such, we can bring the arguments on which interargument relations need to be expressed to the level of the arguments of the constraints of the program.

This is not a difficult transformation to perform. The problem, however, is to decide how much of the program is flattened as the complexity of the analysis is influenced by this choice. Providing an answer to this question requires further investigation.

9.1.3 Interargument relations with multiple conjuncts

Some programs require for certain predicates multiple polynomial inequalities to represent the necessary interargument relations for proving program termination, e.g., the *permutations* program of the Evaluation section of Chapter 8. Another example of such a program is discussed in Section 6.5:

$$a(X,Y), a(Y,Z) \Rightarrow a(X,Z).$$

In [PDS09a], we prove termination of this rule automatically. However, in [PDS09a], to be able to do so, we were required to represent the call set of the $a/2$ constraints, in contrast to Definition 6.1.7, by interargument relations that are the conjunction of linear polynomial inequalities.

To see this, consider the following call set relation: $R^{CS}_{a/2} = \{a(t_1,t_2) \mid t_1,t_2 \in Term_P \wedge \|t_2\|_{ts} \succ_{\mathbb{Z}} \|t_1\|_{ts}\}$. Based on this call set relation, we cannot derive a measure for the size of $a/2$ constraints to prove termination. To prove termination, we additionally need to consider that the size of the nodes is bounded from above. That is, there is some maximally sized node in the input graph. Say, we represent this node by max. Then, subtracting from the term-size of max the difference in term-size for the arguments, i.e., $|a(t_1,t_2)|_{\mathbb{P}} = \|max\|_{ts} - (\|t_2\|_{ts} - \|t_1\|_{ts})$, we can verify a decrease between the heads and the body of the rule. Unfortunately, to allow for such a level mapping, we need an interpretation $\mathbb{P}(a/2) = \|max\|_{ts} - (X - Y)$ that can only be verified to be $\mathbb{N}$-closed if the call set is given by a call set relation: $R^{CS}_{a/2} = \{a(t_1,t_2) \mid t_1,t_2 \in Term_P \wedge \|t_2\|_{ts} \succ_{\mathbb{Z}} \|t_1\|_{ts} \wedge \|max\|_{ts} \succ_{\mathbb{Z}} \|t_1\|_{ts}\}$.

Extending the approach of Chapter 6, to allow for such polynomial interargument relations, is not difficult. We only need to allow for multiple call set relations, in their current form, using for each of these call set relations different identifiers. Then, instead of verifying that some constraint belongs to its call set relation, it has to be verified to belong to several call set relations. The problem, however, is to decide how many call set relations one would include for a certain predicate as the complexity of the analysis is influenced by this choice. Providing an answer to this question requires further investigation.

9.1.4 Limitations not specific to CHR

There are many more problems regarding the automation of termination proofs. Most of these problems are not specific to CHR. We briefly discuss four problems that are interesting to resolve from a practical point of view.

In certain cases, to prove termination, calls to the same predicate require *different success set relations*. In these cases, it would be beneficial to be able to handle them separately. For similar purposes, in [SBG08] a *transitive multi-variant specialisation* is applied. That is, for every call to a predicate, a copy of the implementation of that predicate is introduced and the call to the predicate is adapted appropriately.

A second problem is *different uses of functor symbols* within the context of a single program. For termination analysis of such programs, different interpretations might be required for these functor symbols. To tackle this problem, in [NBDSL06] a *renaming transformation* has been proposed, that has proven to be successful on many programs in practice [NDSGSK11].

A third important problem is *the handling of program constants*. Whenever a program contains constants, it might be difficult to prove termination of these programs. Consider for example a CHR(Prolog) program

$$a(X) \Leftrightarrow less(X, s(s(s(s(s(s(s(s(s(0)))))))))) \mid a(s(X)).$$

It is a bounded increase problem that can only be proven terminating if we consider 4-bit representations to represent the constant of the program. Since program constants can be arbitrary large, it would therefore be interesting to consider an alternative approach to avoid the necessity of ever larger bit representations.

One solution is to transform programs with constants to more general programs (without constants), in which the value of constants is query-based and passed around in extra arguments of the predicates of the program. Considering the

example program from above, we can transform it to the more general program

$$a(X,Y) \Leftrightarrow less(X,Y) \mid a(s(X),Y).$$

Here, the constant is an arbitrary value Y from the query of the program. Therefore, termination of this program implies termination of the original program. Note that we can prove termination of this program, using only 1-bit representations (see *constrbinc*01 of Table 8.4).

A fourth problem is related to *the requirement of a lexicographical ordering* on the constraints of the program, as we need to prove termination of non-primitive recursion. As in [NGSKDS08, SKGN10], to handle such programs, the approach of Chapter 6 needs to be extended with the underlying principles of the dependency pair approach [AG00].

The adaption of these principles is straightforward. That is, to obtain a lexicographical ordering on constraints, the decrease conditions for termination of a program are "relaxed" and then iteratively solved. In [NGSKDS08], this principle has been successfully integrated in LP termination analysis using a constraint-based approach. Similarly, such an integration is also possible for CHR termination analysis.

9.2 Self-sustainability of propagation

Finally, it would be useful to have a characterisation of self-sustainability of propagation considering a fire-once policy for propagation rules. Intuitively, under such a policy, if no new combinations of constraints on which the propagation rule can fire are ever introduced as a result of the application of the propagation rule, then the propagation rule is not self-sustainable. This property of a propagation rule can be related to a CHR net. That is, we can verify whether there are no cycles in the CHR net, across the propagation rule, involving an added CHR constraint of the program. If so, the propagation rule can be ignored since by addition of its constraints, it cannot introduce new combinations of constraints that can fire the propagation rule.

Note that in that case the propagation rule can be removed from the SCC to which it belongs. Therefore, if we obtain by this approach a SCC that ultimately does not contain propagation, we can still verify whether the SCC is non-self-sustainable by the approach of Chapter 7.

Chapter 10

Conclusions

The goal of this thesis: *Termination analysis of CHR: Theory and practice*; was to study the termination behaviour of CHR programs. To this end, similar to the study of termination analysis in other programming languages, we proceeded along the following two lines:

- Provide theoretical conditions, (possibly necessary and) sufficient for termination. These conditions provide a better theoretical understanding of the termination problem, but cannot be verified automatically.
- Provide verifiable conditions, sufficient for termination. These conditions serve as a basis for verification tools and are derived on the basis of theoretical termination conditions.

The contributions of this thesis are therefore twofold. We provided for a theoretical framework for termination analysis of the full CHR language. Based on this theoretical framework, we derived a modularised approach for automated termination analysis of CHR. Finally, this work resulted in T*CoP, a Termination analyser for CHR on top of Prolog, that works well in practice.

10.1 A formal framework for CHR-termination

In the first part of this thesis, we worked towards a formal framework for termination analysis of CHR, providing an in-depth discussion of termination

behaviour of CHR programs. Currently, this is the only framework for termination analysis of the full CHR language.

The essence of our approach is in the understanding that propagation serves as a way to "complete" the information contained within the constraint store. By characterising the effect of full propagation using a propagation store, we obtained a new CHR state representation that can be seen to decrease for terminating CHR programs with any kind of rule. Then, we provided for conditions, i.e., the RC for CHR, that guarantees decreases on this new CHR state representation.

Furthermore, we gained the understanding that the termination problem for CHR can be viewed separately for propagation and for simpagation w.r.t. propagation. That is, for termination, propagation rules must never introduce constraints that through further propagation can replace the head constraints that gave cause to these constraints. On the other hand, simpagation rules must demonstrate a strict multiset decrease between the removed head and body of the rule, such that by further propagation on the added constraints of the simpagation rule this multiset decrease cannot be undone.

Future work, regarding formal frameworks, need to overcome a number of shortcomings of the RC for CHR (see Chapter 5). First, perhaps most importantly, we need to overcome the problem of single-headed propagation rules. For such rules the RC for CHR is not sufficiently precise. Secondly, we need to consider the success set of CHR predicates. Finally, we need to consider *necessity* of termination conditions. Although this does not guarantee practical relevance of termination conditions, it does provide us with an insight into the minimal requirements for termination of a CHR program.

10.2 Verifiable conditions for CHR-termination

In the second part of this thesis, we developed verifiable conditions on the basis of integer polynomial interpretations. Then, we automated the search for a polynomial interpretation by a constraint-based approach. Although our approach is strongly related to the techniques with polynomial interpretations in LP, it is new in a sense that it combines integer polynomials and the derivation of call set relations within a constraint-based approach, which has been extended to handle multiset decreases.

Since, in practice, a proof of termination is more efficiently obtained if the proof is modularised, we adapted the approach based on SCCs in LP to CHR. Furthermore, we provided a characterisation of CHR cycles on the basis of self-

sustainability. This characterisation, which is a rather good approximation of cyclic behaviour in CHR programs, differs significantly from such a notion in LP. Furthermore, it can be used to formulate non-self-sustainability of a SCC, providing us with an additional approach for proving part of a CHR program terminating.

Future work, regarding automation of termination analysis for CHR, need to overcome a number of shortcomings as well (see Chapter 9). First of all, we need to improve interargument relations to consider multiple conjunctive polynomial inequalities as there are many practical programs for which such a refinement is required to prove termination. Secondly, we need to formalise the notion of self-sustainability of propagation as it will provide us with an efficient test to ignore propagation rules in SCCs. Thirdly, we need to implement the underlying principles of the dependency-pair approach to be able to construct a lexicographical ordering on the constraints of the program, e.g., to handle non-primitive recursion. Finally, we need to investigate the benefit of program transformations such as renaming, flattening, single-fact specialisation, transitive multi-variant specialisation, unfolding, etc. to improve precision of the termination analysis. In this context, it would be interesting to work towards a standard form for termination analysis of CHR programs.

Bibliography

[AB90] Krzysztof R. Apt and Marc Bezem. Acyclic programs. In *ICLP '90: Proceedings of the 7th International Conference on Logic Programming*, pages 617–633, 1990. pages 1, 3

[Abd97] Slim Abdennadher. Operational semantics and confluence of Constraint Propagation Rules. In *CP '97: Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming*, pages 252–266, 1997. pages 3, 11, 44, 46, 47

[AF99] Slim Abdennadher and Thom W. Frühwirth. Operational equivalence of CHR programs and constraints. In *CP '99: Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming*, pages 43–57, 1999. pages 3

[AFM96] Slim Abdennadher, Thom W. Frühwirth, and Holger Meuss. On confluence of Constraint Handling Rules. In *CP '96: Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming*, pages 1–15, 1996. pages 3

[AG00] Thomas Arts and Jürgen Giesl. Termination of Term Rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000. pages 174

[AP91] Krzysztof R. Apt and Dino Pedreschi. Proving termination of general Prolog programs. In *TACS '91: Proceedings of the International Conference on Theoretical Aspects of Computer Software*, pages 265–289, 1991. pages 1, 3, 11

[AP94] Krzysztof R. Apt and Dino Pedreschi. *Advances in logic programming theory*, chapter Modular termination proofs for

Logic and pure Prolog programs, pages 183–229. Oxford University Press, 1994. pages 1, 3

[Apt90] Krzysztof R. Apt. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, chapter Logic Programming, pages 493–574. Springer-Verlag, 1990. pages 2, 15

[BCER02] Annalisa Bossi, Nicoletta Cocco, Sandro Etalle, and Sabina Rossi. On modular termination proofs of general Logic Programs. *Theory and Practice of Logic Programming*, 2(3):263–291, 2002. pages 1, 3, 95

[BCF91] Annalisa Bossi, Nicoletta Cocco, and Massimo Fabris. Proving termination of Logic Programs by exploiting term properties. In *TAPSOFT '91: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Volume 2: Advances in Distributed Computing (ADC) and Colloquium on Combining Paradigms for Software Developmemnt (CCPSD)*, pages 153–180, 1991. pages 3, 29, 30

[BCG+07] Maurice Bruynooghe, Michael Codish, John P. Gallagher, Samir Genaim, and Wim Vanhoof. Termination analysis of Logic Programs through combination of type-based norms. *ACM Transactions on Programming Languages and Systems*, 29(2):1–44, 2007. pages 1, 3, 11, 102, 156

[BCLM88] Jean-Pierre Banâtre, A. Coutant, and D. Le Metayer. A parallel machine for multiset transformation and its programming style. *Future Generation Computer Systems*, 4(2):133–144, 1988. pages 2

[Bea96] J. E. Beasley, editor. *Advances in linear and integer programming*. Oxford University Press, 1996. pages 158

[Bli89] Wayne D. Blizard. Multiset theory. *Notre Dame Journal of Formal Logic*, 30(1):36–66, 1989. pages 15

[BLNM+09] Cristina Borralleras, Salvador Lucas, Rafael Navarro-Marset, Enric Rodríguez-Carbonell, and Albert Rubio. Solving non-linear polynomial arithmetic via SAT modulo linear arithmetic. In *CADE '09: Proceedings of the 22nd International Conference on Automated Deduction*, pages 294–305, 2009. pages 143

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and all that*. Cambridge University Press, 1998. pages 2

[CGBA+11] Michael Codish, Igor Gonopolskiy, Amir M. Ben-Amram, Carsten Fuhs, and Jürgen Giesl. SAT-based termination analysis using monotonicity constraints over the integers. *Theory and Practice of Logic Programming*, 11(4-5):503–520, 2011. pages 144, 163

[CMTU05] Evelyne Contejean, Claude Marché, Ana Paula Tomás, and Xavier Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005. pages 143, 144

[CT99] Michael Codish and Cohavit Taboch. A semantic basis for the termination analysis of Logic Programs. *Journal of Logic Programming*, 41(1):103–123, 1999. pages 1, 3

[DDSV99] Stefaan Decorte, Danny De Schreye, and Henk Vandecasteele. Constraint-based termination analysis of Logic Programs. *ACM Transactions on Programming Languages and Systems*, 21(6):1137–1195, 1999. pages 1, 3, 4, 11, 102, 103, 125, 146, 170

[Der82] Nachum Dershowitz. Orderings for Term-Rewriting Systems. *Theoretical Computer Science*, 17:279–301, 1982. pages 23, 26, 71, 76

[Der87] Nachum Dershowitz. Termination of Rewriting. *Journal of Symbolic Computing*, 3(1/2):69–116, 1987. pages 1, 3

[DK08] Leslie De Koninck. *Execution Control for Constraint Handling Rules*. PhD thesis, Katholieke Universiteit Leuven - Departement Computer Wetenschappen, Belgium, 2008. pages 11

[DK09] Leslie De Koninck. Logical Algorithms meets CHR: A meta-complexity result for Constraint Handling Rules with rule priorities. *Theory and Practice of Logic Programming*, 9(2):165–212, 2009. pages 99

[DLSS01] Nachum Dershowitz, Naomi Lindenstrauss, Yehoshua Sagiv, and Alexander Serebrenik. A general framework for automatic termination analysis of Logic Programs. *Applicable Algebra in Engineering, Communication and Computing*, 12(1/2):117–156, 2001. pages 1

[DM79] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979. pages 1, 3, 15, 23, 26, 71, 76

[DSD94] Danny De Schreye and Stefaan Decorte. Termination of Logic Programs: The never-ending story. *Journal of Logic Programming*, 19/20:199–260, 1994. pages 1, 3, 4, 11, 22, 34, 102

[DSGH04] Gregory J. Duck, Peter J. Stuckey, María García de la Banda, and Christian Holzbaur. The refined operational semantics of Constraint Handling Rules. In *ICLP '04: Proceedings of the 20th International Conference on Logic Programming*, pages 90–104, 2004. pages 11, 99

[DSS02] Danny De Schreye and Alexander Serebrenik. Acceptability with general orderings. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*, pages 187–210. Springer-Verlag, 2002. pages 1, 4, 11, 24, 29, 102, 110

[EWZ08] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of Term Rewriting. *Journal of Automated Reasoning*, 40(2-3):195–220, 2008. pages 1, 3, 11, 102

[Far02] Julius G. Farkas. Über die Theorie der Einfachen Ungleichungen. *Journal für die Reine und Angewandte Mathematik*, 124:1–27, 1902. pages 157

[FGM+07] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. SAT solving for termination analysis with polynomial interpretations. In *SAT '07: Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing*, pages 340–354, 2007. pages 144

[Frü98] Thom W. Frühwirth. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1-3):95–138, 1998. pages 2, 5, 15, 16, 18, 32, 44

[Frü00] Thom W. Frühwirth. Proving termination of Constraint Solver Programs. In *Selected papers from the Joint ERCIM/Compulog Net Workshop on New Trends in Contraints*, pages 298–317. Springer-Verlag, 2000. pages 1, 7, 10, 12, 14, 30, 90, 95, 145, 159, 171

[GC05] Samir Genaim and Michael Codish. Inferring termination conditions for Logic Programs using backwards analysis. *Theory*

and Practice of Logic Programming, 5(1-2):75–91, 2005. pages 1, 3, 11, 102

[GSKT+07] Jürgen Giesl, Peter Schneider-Kamp, René Thiemann, Stephan Swiderski, Manh Thang Nguyen, Danny De Schreye, and Alexander Serebrenik. Termination of programs using Term Rewriting and SAT solving. In *Deduction and Decision Procedures*, 2007. pages 1, 3, 11, 102

[GTSKF04] Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Automated Termination Proofs with AProVE. In *RTA '04: Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, pages 210–220, 2004. pages 1, 3, 5, 11, 102, 103

[HJ98] Hoon Hong and Dalibor Jakus. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21(1):23–38, 1998. pages 142

[JB92] Gerda Janssens and Maurice Bruynooghe. Deriving descriptions of possible values of program variables by means of abstract interpretation. *Journal of Logic Programming*, 13(2-3):199–260, 1992. pages 119, 123, 134

[JM94] Joxan Jaffar and Michael J. Maher. Constraint Logic Programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994. pages 2

[JMMS98] Joxan Jaffar, Michael J. Maher, Kim Marriot, and Peter J. Stuckey. The semantics of Constraint Logic Programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998. pages 2

[Kow79] Robert A. Kowalski. Algorithm = Logic + Control. *Communications of the ACM*, 22(7):424–436, 1979. pages 1

[Lan79] D.S. Lankford. On proving term rewriting systems are noetherian. Technical report, Louisiana Technological University, Ruston, LA, USA, 1979. pages 103

[LJBA01] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In *POPL '01: Proceedings of the 28th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages*, pages 81–92, 2001. pages 1, 3

[Llo87] John W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987. pages 2, 15, 31, 118

[LSS97] Naomi Lindenstrauss, Yehoshua Sagiv, and Alexander Serebrenik. TermiLog: A system for checking termination of queries to Logic Programs. In *CAV '97: Proceedings of the 9th International Conference on Computer Aided Verification*, pages 444–447, 1997. pages 1, 3, 103

[MB05] Frédéric Mesnard and Roberto Bagnara. cTI: A constraint-based termination inference tool for ISO-Prolog. *Theory and Practice of Logic Programming*, 5(1-2):243–257, 2005. pages 1, 3, 11, 102, 103, 131

[Min10] MiniSAT, 2010. `http://minisat.se/`. pages 163

[MR03] Frédéric Mesnard and Salvatore Ruggieri. On proving left termination of Constraint Logic Programs. *ACM Transactions on Computational Logic*, 4(2):1–26, 2003. pages 3

[MS98] Kim Marriott and P. J. Stuckey. *Programming with constraints: an introduction*. MIT Press, 1998. pages 2

[NBDSL06] Manh Thang Nguyen, Maurice Bruynooghe, Danny De Schreye, and Michael Leuschel. Program specialisation as a preprocessing step for termination analysis. In *WST '06: Proceedings of the 8th International Workshop on Termination*, pages 7–11, 2006. pages 173

[NDSGSK11] Manh Thang Nguyen, Danny De Schreye, Jürgen Giesl, and Peter Schneider-Kamp. Polytool: Polynomial interpretations as a basis for termination analysis of Logic Programs. *Theory and Practice of Logic Programming*, 11(1):33–63, 2011. pages 1, 3, 4, 5, 11, 12, 24, 95, 102, 103, 104, 105, 106, 107, 108, 110, 111, 115, 117, 118, 119, 123, 124, 125, 126, 127, 128, 134, 135, 141, 142, 161, 170, 173

[NGSKDS08] Manh Thang Nguyen, Jürgen Giesl, Peter Schneider-Kamp, and Danny De Schreye. Termination analysis of Logic Programs based on dependency graphs. In *LNCS 4915: Proceedings of the 17th International Symposium on Logic-Based Program Synthesis and Transformation, Revised Selected Papers*, pages 8–22. Springer Verlag, 2008. pages 24, 146, 148, 174

[Ngu09] Manh Thang Nguyen. *Termination Analysis: Crossing Paradigm Borders*. PhD thesis, Katholieke Universiteit Leuven - Departement Computer Wetenschappen, Belgium, 2009. pages 28, 119

[OCM00] Enno Ohlebusch, Claus Claves, and Claude Marché. TALP: A tool for the termination analysis of Logic Programs. In *RTA '00: Proceedings of the 11th International Conference on Rewriting Techniques and Applications*, pages 270–273, 2000. pages 1, 3

[PDS08a] Paolo Pilozzi and Danny De Schreye. Termination analysis of CHR revisited. In *ICLP '08: Proceedings of the 24th International Conference on Logic Programming*, pages 501–515, 2008. pages 10, 12, 62, 91

[PDS08b] Paolo Pilozzi and Danny De Schreye. Termination analysis of CHR revisited. In *CHR '08: Proceedings of the 5th International Workshop on Constraint Handling Rules*, pages 35–50, 2008. pages 45

[PDS09a] Paolo Pilozzi and Danny De Schreye. Automating termination proofs for CHR. In *ICLP '09: Proceedings of the 25th International Conference on Logic Programming*, pages 504–508, 2009. pages 28, 104, 105, 119, 123, 141, 145, 170, 172

[PDS09b] Paolo Pilozzi and Danny De Schreye. Proving termination by invariance relations. In *ICLP '09: Proceedings of the 25th International Conference on Logic Programming*, pages 499–503, 2009. pages 104

[PDS09c] Paolo Pilozzi and Danny De Schreye. Scaling termination proofs by a characterization of cycles in CHR. Technical Report CW 541, Katholieke Universiteit Leuven - Departement Computer Wetenschappen, Leuven, Belgium, 2009. pages 155

[PDS10] Paolo Pilozzi and Danny De Schreye. Scaling termination proofs by a characterisation of cycles in CHR. In *WST '07: Proceedings of the 11th International Workshop on Termination*, page 5, 2010. pages 148, 155

[PM09] Étienne Payet and Frédéric Mesnard. A non-termination criterion for binary Constraint Logic Programs. *Theory and Practice of Logic Programming*, 9(2):145–164, 2009. pages 3

[PSB10] Paolo Pilozzi, Tom Schrijvers, and Maurice Bruynooghe. A transformational approach for proving properties of the CHR constraint store. In *LNCS 6037: Proceedings of the 19th International Symposium on Logic-Based Program Synthesis and Transformation, Revised Selected Papers*, pages 22–36. Springer Verlag, 2010. pages 123

[PSDS07a] Paolo Pilozzi, Tom Schrijvers, and Danny De Schreye. Proving termination of CHR in Prolog: A transformational approach. Technical Report CW 487, Katholieke Universiteit Leuven - Departement Computer Wetenschappen, Leuven, Belgium, 2007. pages 5, 95, 96, 97, 159, 171

[PSDS07b] Paolo Pilozzi, Tom Schrijvers, and Danny De Schreye. Proving termination of CHR in Prolog: A transformational approach. In *WST '07: Proceedings of the 9th International Workshop on Termination*, pages 30–33, 2007. pages 5, 95, 96, 97, 159, 171

[Ram30] Frank P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical society (2)*, 30:264–268, 1930. pages 156

[SBG08] Tom Schrijvers, Maurice Bruynooghe, and John P. Gallagher. From monomorphic to polymorphic well-typings and beyond. In *LOPSTR '08: Pre-Proceedings of the 18th International Symposium on Logic-Based Program Synthesis and Transformation*, pages 152–167, 2008. pages 173

[Sch86] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1986. pages 148, 155

[Sch05] Tom Schrijvers. *Analyses, Optimizations and Extensions of Constraint Handling Rules.* PhD thesis, Katholieke Universiteit Leuven - Departement Computer Wetenschappen, Belgium, 2005. pages 3, 11, 44, 46, 62

[Sch08] Tom Schrijvers. Constraint Handling Rules – A tutorial for (Prolog) programmers. In *ICLP '08: Proceedings of the 24th International Conference on Logic Programming*, pages 9–10, 2008. pages 2

[SDS00] Alexander Serebrenik and Danny De Schreye. Termination analysis of Logic Programs using acceptability with general term orders. *Computing Research Repository*, cs.PL/0011025, 2000. pages 23

[SDS01] Alexander Serebrenik and Danny De Schreye. Non-transformational termination analysis of Logic Programs, based on general term-orderings. In *LNCS 2042: Proceedings of the 10th International Symposium on Logic-Based Program Synthesis and Transformation, Revised Selected Papers*, pages 69–85. Springer Verlag, 2001. pages 5

[SDS03] Alexander Serebrenik and Danny De Schreye. Hasta-La-Vista: Termination analyser for Logic Programs. In *WLPE '03: Proceedings of the 13th Workshop on Logic Programming Environments*, pages 60–74, 2003. pages 1, 3, 11, 102, 103

[SDS04] Alexander Serebrenik and Danny De Schreye. Inference of termination conditions for numerical loops in Prolog. *Theory and Practice of Logic Programming*, 4(5-6):719–751, 2004. pages 1, 3, 11, 102

[SDS05a] Alexander Serebrenik and Danny De Schreye. On termination of meta-programs. *Theory and Practice of Logic Programming*, 5(3):355–390, 2005. pages 1, 3, 11, 102

[SDS05b] Alexander Serebrenik and Danny De Schreye. Termination of floating-point computations. *Journal of Automated Reasoning*, 34(2):141–177, 2005. pages 1, 3, 11, 102

[SIC11] SICStus, 2011. `http://www.sics.se/sicstus/`. pages 144

[SKGN10] Peter Schneider-Kamp, Jürgen Giesl, and Manh Thang Nguyen. The dependency triple framework for termination of Logic Programs. In *LNCS 6037: Proceedings of the 19th International Symposium on Logic-Based Program Synthesis and Transformation, Revised Selected Papers*, pages 37–51. Springer Verlag, 2010. pages 1, 3, 4, 5, 11, 102, 146, 148, 174

[SKGS+10] Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, and René Thiemann. Automated termination analysis for Logic Programs with cut. *Theory and Practice of Logic Programming*, 10(4-6):365–381, 2010. pages 3

[SKGST06] Peter Schneider-Kamp, Jürgen Giesl, Alexander Serebrenik, and René Thiemann. Automated termination analysis for Logic Programs by Term Rewriting. In *LOPSTR '06: Pre-Proceedings of the 16th International Symposium on Logic-Based Program Synthesis and Transformation*, pages 177–193, 2006. pages 5

[Sne08] Jon Sneyers. *Optimizing Compilation and Computational Complexity of Constraint Handling Rules*. PhD thesis, Katholieke Universiteit Leuven - Departement Computer Wetenschappen, Belgium, 2008. pages 3, 11

[SP10] SWI-Prolog, 2010. `http://www.swi-prolog.org`. pages 161

[SR90] Vijay A. Saraswat and Martin C. Rinard. Concurrent Constraint Programming. In *POPL '90: Proceedings of the 17th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages*, pages 232–245, 1990. pages 2

[SS96] Wayne Snyder and James G. Schmolze. Rewrite semantics for Production Rule Systems: Theory and applications. In *CADE '96: Proceedings of the 13th International Conference on Automated Deduction*, pages 508–522, 1996. pages 2

[SSD05] Tom Schrijvers, Peter J. Stuckey, and Gregory J. Duck. Abstract interpretation for Constraint Handling Rules. In *PPDP '05: Proceedings of the 7th International Conference on Principles and Practices of Declarative Programming*, pages 218–229, 2005. pages 3

[Stä98] Robert F. Stärk. The theoretical foundations of LPTP (A Logic Program Theorem Prover). *Journal of Logic Programming*, 36(3):241–269, 1998. pages 1, 3

[SVWSDK10] Jon Sneyers, Peter Van Weert, Tom Schrijvers, and Leslie De Koninck. As time goes by: Constraint Handling Rules – A survey of CHR research between 1998 and 2007. *Theory and Practice of Logic Programming*, 10(1):1–47, 2010. pages 2, 3, 10, 11, 98

[Tar72] Robert E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. pages 151, 162

[T*C10] T*CoP, 2010. `http://people.cs.kuleuven.be/~paolo.pilozzi?pg=tcop`. pages 160, 163

[Ter10] Term Rewriting and Logic Programming Termination Analysis Competition, 2010. `http://termination-portal.org/`. pages 103, 146, 166

[VDS01] Sofie Verbaeten and Danny De Schreye. Termination of simply-moded well-typed Logic Programs under a tabled execution mechanism. *Applicable Algebra in Engineering, Communication and Computing*, 12(1/2):157–196, 2001. pages 3

[VDSP08] Dean Voets, Danny De Schreye, and Paolo Pilozzi. A new approach to termination analysis of Constraint Handling Rules. In *LOPSTR '08: Pre-Proceedings of the 18th*

International Symposium on Logic-Based Program Synthesis and Transformation, pages 28–42, 2008. pages 10, 12, 44, 45, 51, 52, 53, 90, 162

[VPDS07] Dean Voets, Paolo Pilozzi, and Danny De Schreye. A new approach to termination analysis of Constraint Handling Rules. In *CHR '07: Proceedings of the 4th International Workshop on Constraint Handling Rules*, pages 77–89, 2007. pages 52

[Wal07] Johannes Waldmann. Weighted automata for proving termination of String Rewriting. *Journal of Automata, Languages and Combinatorics*, 12(4):545–570, 2007. pages 3

Biography

Paolo Pilozzi was born on the 13th of June 1981 in Leuven, Belgium. He grew up in different cities around Belgium ending up again in Leuven. There, he graduated from the Heilig-Drievuldigheids College Leuven in 1999. He studied industrial engineering at Groep T in Leuven, where he graduated cum laude in 2005. His master's thesis was titled "Hardware acceleration of LabVIEW applications with the Virtex-II Pro Development Kit." and supervised by Prof. Dr. Luc Bienstman. Afterwards, he did a Master of Artificial Intelligence at the Katholieke Universiteit Leuven, where he graduated magna cum laude in 2006. His master's thesis was titled "Modularized learning of genetic networks." and supervised by Prof. Dr. ir. Bart De Moor.

In October 2006, Paolo started working as a Ph.D. student at the Department of Computer Science of the Katholieke Universiteit Leuven, where he joined the Analysis subgroup of the DTAI (Declarative Languages and Artificial Intelligence) research group to work on Termination Analysis of Constraint Handling Rules. His work was supervised by Prof. Dr. Daniel De Schreye. Before 2008, Paolo was funded by the Fund for Scientific Research (FWO) in Flanders (Belgium). Since January 2008, Paolo is funded by the Institute for Science and Technology (IWT) in Flanders (Belgium). In 2007 and 2010, Paolo was part of the winning team of the 14th and 17th Prolog Programming Contest at ICLP'07 in Porto (Portugal) and at ICLP'10 in Edinburgh (U.K.).

List of Publications

Publications at International Conferences

Pilozzi, Paolo; De Schreye, Danny. Improved termination analysis of CHR using self-sustainability analysis. 21st International Symposium on Logic-Based Program Synthesis and Transformation. Odense, Denmark, July 2011. Revised Selected Papers (accepted).

Pilozzi, Paolo; De Schreye, Danny. Improved termination analysis of CHR using self-sustainability analysis. 21st International Symposium on Logic-Based Program Synthesis and Transformation. Odense, Denmark, July 2011. Pre-proceedings, pages 214-228.

Pilozzi, Paolo; Schrijvers, Tom; Bruynooghe, Maurice. A transformational approach for proving properties of the CHR constraint store. 19th International Symposium on Logic-Based Program Synthesis and Transformation. Coimbra, Portugal, September 2009. Revised Selected Papers, volume 6037 of Lecture Notes in Computer Science, pages 22-36, Springer-Verlag.

Pilozzi, Paolo. Research summary: Termination of CHR. 25th International Conference on Logic Programming. Pasadena, California, U.S.A., 14-17 July 2009. Proceedings, volume 5649 of Lecture Notes in Computer Science, pages 534-535, Springer-Verlag.

Pilozzi, Paolo; De Schreye, Danny. Automating termination proofs for CHR. 25th International Conference on Logic Programming. Pasadena, California, U.S.A., 14-17 July 2009. Proceedings, volume 5649 of Lecture Notes in Computer Science, pages 504-508, Springer-Verlag.

Pilozzi, Paolo; De Schreye, Danny. Proving termination by invariance relations. 25th International Conference on Logic Programming. Pasadena,

California, U.S.A., 14-17 July 2009. Proceedings, volume 5649 of Lecture Notes in Computer Science, pages 499-503, Springer-Verlag.

Pilozzi, Paolo; Schrijvers, Tom; Bruynooghe, Maurice. A transformational approach for proving properties of the CHR constraint store. 19th International Symposium on Logic-Based Program Synthesis and Transformation. Coimbra, Portugal, 9-11 September 2009. Pre-proceedings, pages 20-29.

Pilozzi, Paolo; De Schreye, Danny. Termination analysis of CHR revisited. 24th International Conference on Logic Programming. Udine, Italy, 9-13 December 2008. Proceedings, volume 5366 of Lecture Notes in Computer Science, pages 501-515, Springer-Verlag.

Voets, Dean; Pilozzi, Paolo; De Schreye, Danny. A new approach to termination analysis of Constraint Handling Rules. 18th International Symposium on Logic-Based Program Synthesis and Transformation. Valencia, Spain, 17-18 July 2008. Pre-Proceedings, pages 28-42.

Publications at International Workshops

Pilozzi, Paolo; De Schreye, Danny. Scaling termination proofs by a characterisation of cycles in CHR. 11th International Workshop on Termination. Edinburgh, United Kingdom, 14-15 July 2010, Proceedings, 5 pages.

Pilozzi, Paolo; De Schreye, Danny. Termination analysis of CHR revisited. 5th International Workshop on Constraint Handling Rules. Hagenberg, Austria, 14 July 2008. Proceedings, pages 1-16, RISC Report Series 08-10.

Voets, Dean; Pilozzi, Paolo; De Schreye, Danny. A new approach to termination analysis of Constraint Handling Rules. 4th International Workshop on Constraint Handling Rules. Porto, Portugal, 8 September 2007. Proceedings, pages 77-89.

Pilozzi, Paolo; Schrijvers, Tom; De Schreye, Danny. Proving termination of CHR in Prolog: A transformational approach. 9th International Workshop on Termination. Paris, France, 29 June 2007. Proceedings, pages 30-33.

Voets, Dean; Pilozzi, Paolo; De Schreye, Danny. A new approach to termination analysis of Constraint Handling Rules. 9th International

Workshop on Termination. Paris, France, 29 June 2007. Proceedings, pages 26-29.

Poster presentations

Pilozzi, Paolo; De Schreye, Danny. Termination Analysis of CHR. WOG Seminar on Logic and Computation. Oostduinkerke, Belgium, 28-29 April 2008.

Technical Reports

Pilozzi, Paolo; De Schreye, Danny. Scaling termination proofs by a characterization of cycles in CHR, Department of Computer Science. K.U.Leuven, Report CW 541. Leuven, Belgium, April 2009.

Voets, Dean; Pilozzi, Paolo; De Schreye, Danny. A new approach to termination analysis of CHR. Department of Computer Science, K.U.Leuven, Report CW 506. Leuven, Belgium, January 2008.

Pilozzi, Paolo; Schrijvers, Tom; De Schreye, Danny. Proving termination of CHR in Prolog: A transformational approach. Department of Computer Science, K.U.Leuven, Report CW 487. Leuven, Belgium, April 2007.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science
Declarative Languages and Artificial Intelligence
Celestijnenlaan 200A
B-3001 Heverlee